UNCLASSIFIED

IDA PAPER P-2124

A PORTABLE Ada MULTITASKING ANALYSIS SYSTEM

Robert J. Knapper
David O. LeVan

December 1988

Prepared for
STARS Joint Program Office

INSTITUTE FOR DEFENSE ANALYSES
1801 N. Beauregard Street, Alexandria, Virginia 22311-1772

UNCLASSIFIED

IDA Log No. HQ 88-033512

DEFINITIONS

IDA publishes the following documents to report the results of its work.

Reports

Reports are the most authoritative and most carefully considered products IDA publishes. They normally embody results of major projects which (a) have a direct bearing on decisions affecting major programs, (b) address issues of significant concern to the Executive Branch, the Congress and/or the public, or (c) address issues that have significant economic implications. IDA Reports are reviewed by outside panels of experts to ensure their high quality and relevance to the problems studied, and they are released by the President of IDA.

Group Reports

Group Reports record the findings and results of IDA established working groups and panels composed of senior individuals addressing major issues which otherwise would be the subject of an IDA Report. IDA Group Reports are reviewed by the senior individuals responsible for the project and others as selected by IDA to ensure their high quality and relevance to the problems studied, and are released by the President of IDA.

Papers

Papers, also authoritative and carefully considered products of IDA, address studies that are narrower in scope than those covered in Reports. IDA Papers are reviewed to ensure that they meet the high standards expected of refereed papers in professional journals or formal Agency reports.

Documents

IDA Documents are used for the convenience of the sponsors or the analysts (a) to record substantive work done in quick reaction studies, (b) to record the proceedings of conferences and meetings, (c) to make available preliminary and tentative results of analyses, (d) to record data developed in the course of an investigation, or (e) to forward information that is essentially unanalyzed and unevaluated. The review of IDA Documents is suited to their content and intended use.

The work reported in this document was conducted under contract MDA 903 84 C 0031 for the Department of Defense. The publication of this IDA document does not indicate endorsement by the Department of Defense, nor should the contents be construed as reflecting the official position of that Agency.

This Paper has been reviewed by IDA to assure that it meets high standards of thoroughness, objectivity, and appropriate analytical methodology and that the results, conclusions and recommendations are properly supported by the material presented.

Approved for public release, unlimited distribution; 05 September 1990. Unclassified.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	December 1988	Final

4. TITLE AND SUBTITLE
A Portable Ada Multitasking Analysis System

5. FUNDING NUMBERS
MDA 903 84 C 0031
A-134

6. AUTHOR(S)
Robert J. Knapper, David O. LeVan

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Institute for Defense Analyses
1801 N. Beauregard St.
Alexandria, VA 22311-1772

8. PERFORMING ORGANIZATION REPORT NUMBER
IDA Paper P-2124

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
STARS Joint Program Office
1400 Wilson Blvd.
Arlington, VA 22209-2308

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release, unlimited distribution; 05 September 1990.

12b. DISTRIBUTION CODE
2A

13. ABSTRACT *(Maximum 200 words)*

IDA Paper P-2124, A Portable Ada Multitasking Analysis System, documents the design and implementation of a prototype software tool to assist in the dynamic analysis of Ada multitasking programs. This document will be used as a reference and guide for Ada tasking analysis using the prototype tool and will provide a basis from which future research can proceed.

14. SUBJECT TERMS
Ada Multitasking; Prorotyping; Software Engineering Environment (SEE); Portable Ada Multitasking System (PAMAS); Debuggers; Analyzer Tool.

15. NUMBER OF PAGES
202

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

IDA PAPER P-2124

A PORTABLE Ada MULTITASKING ANALYSIS SYSTEM

Robert J. Knapper
David O. LeVan

December 1988

INSTITUTE FOR DEFENSE ANALYSES

Contract MDA 903 84 C 0031
DARPA Assignment A-134

TABLE OF CONTENTS

PREFACE

The purpose of IDA Paper P-2124, *A Portable Ada Multitasking Analysis System,* is to document the design and implementation of a prototype software tool to assist in the dynamic analysis of Ada multitasking programs.

The importance of this document is based on fulfilling the objective of Task Order T-D5-429, Software Technology Acceleration Project, which is to "perform technical assessments of strategies for accelerating the development of Ada-related software technologies" by "prototyping Ada software development." P-2124 will be used as a reference and guide for Ada tasking analysis using the prototype tool and will provide a basis from which future research can proceed. As a paper, P-2124 is directed towards individuals interested in developing more predictable Ada software.

The document was reviewed by the members of the following CSED Peer Review: Mr. Nelson Corcoran, Dr. Dennis Fife, Mr Terry Mayfield, Dr. Reginald Meeson, and Ms. Sylvia Reynolds.

A Portable Ada Multitasking Analysis System

1. INTRODUCTION

This paper describes the design and implementation of a prototype, system independent tool for analyzing Ada programs with multiple tasks. It is not a debugger per se, however, the tool does enable the user to monitor and to some extent control task interrelationships dynamically. The tool operates at an Ada source level and can be ported to virtually any system supporting Ada. This not only allows the tool to be widely applicable, but also provides a mechanism to test identical software for consistency of execution across multiple systems. The Software Technology for Adaptable, Reliable Systems (STARS) Joint Program Office (JPO) is building a Software Engineering Environment (SEE) that will cover the entire software life cycle. Testing and analysis is a critical part of the life cycle. Although this tool is an experimental prototype and only laboratory tested, it does provide a basis from which future research and development in testing and analysis may proceed.

Section 3 provides a background into what Ada tasking is and why having a tool to analyze tasking behavior is useful. The system requirements are specified in Section 4, the system development plan is detailed in Section 5, and the system design specifications are in listed in Section 6. The testing of the system is described in Section 7 and conclusions are in Section 8. Lastly, Appendices A and B contain a user's guide and a listing of the system code.

2. SCOPE

IDA conducted research into the automatic analysis of multitask Ada software over a one year period. The original focus was from the viewpoint that traditional automated debugging techniques could be applied to concurrent software in producing a totally system-independent tool. However, traditional techniques were found to be too system-specific and were discarded in favor of a new approach. This new approach employed Ada tasks as intermediary control structures in establishing and maintaining control over the multitask software being analyzed. The result of the research is the Portable Ada Multitasking Analysis System (PAMAS).

3. BACKGROUND

3.1 Ada Tasking

Real-time and embedded applications usually involve multiple processes executing concurrently. Software for these applications must therefore handle concurrency. Ada uses the **task** to define a concurrent process. The Ada tasking mechanism is provided as a group of language constructs for the creation and synchronization of tasks.

An Ada task has two parts, a specification or interface section and a body or implementation section. The specification section may be null or it may contain **entry** declarations. An entry is a synchronization identifier that is called, very much in the subprogram sense, by another process. A task body is the implementation of the executable statements of a task. If entries are declared in the specification, then the corresponding **accept** blocks are in the body. An accept block is the code that is executed when tasks synchronize or *rendezvous*.

For example, consider the following classic producer-consumer problem. We have a producer producing widgets as fast as possible and we have a consumer consuming those widgets as fast as possible. However, if it takes longer to consume a widget than it does to produce one then it would not be efficient for the producer to wait until the consumer finishes before starting another widget. The producer and consumer are concurrent processes that may be implemented as Ada tasks.

```
task PRODUCER;

task CONSUMER is
   entry RECEIVE (WIDGET : in CONSUMABLE);
end CONSUMER;

task body PRODUCER is
   WIDGET : CONSUMABLE;
begin
   while not FINISHED loop
   PRODUCE (WIDGET);
   CONSUMER.RECEIVE (WIDGET);
   end loop;
end PRODUCER;

task body CONSUMER is
   RECEIVED_WIDGET : CONSUMABLE;
begin
   loop
      accept RECEIVE (WIDGET : in CONSUMABLE) do
         RECEIVED_WIDGET := WIDGET;
      end;
      CONSUME (RECEIVED_WIDGET);
   end loop;
end CONSUMER;
```

In this example, PRODUCER calls the entry RECEIVE and passes to CONSUMER the WIDGET it produced. During the rendezvous, only the code within the accept block in CONSUMER is in execution; PRODUCER's execution is blocked. After receiving the WIDGET, both tasks continue their concurrent execution. CONSUMER proceeds to CONSUME the RECEIVED_WIDGET, while PRODUCER is free to produce another.

Analyzing the example above can be done by manual inspection since it deals with only two tasks and one rendezvous. But as the complexity of the program goes up, we could have a system with hundreds of rendezvous! The analysis of such a system without an automated aid would be tedious and very time consuming.

The next section discusses how the analyzer tool described in this paper differs from the more traditional approach of general purpose debuggers for Ada.

3.2 Debugging and Debuggers

Debugging as a matter of practice has been around since Charles Babbage was constructing his Analytical and Difference Engines in the 19th century, with Ada Lovelace as his programmer. However, this kind of debugging was basically trial and error debugging.

With the advent of the general purpose electronic digital computer, software became more flexible but also more complex. The need to detect and correct bugs increased and the method of manual trial and error debugging was not sufficient. Run-time tools, called debuggers (but are more appropriately debugging aids), provided some automated assistance in detecting and correcting error conditions.

Most debuggers provide for breakpoint setting, variable examination, variable assignment, and traceback information display. This functionality helps to automate the trial and error process and is effective to varying degrees. However, debuggers are almost always run-time specific tools. They have to have access to the run-time kernel in order to provide their functionality. This is particularly true of debuggers for languages that have concurrent constructs, such as Ada.

The Ada Standard does not specify the implementation details for tasking. Therefore, things such as task scheduling, select alternative evaluation order, etc., may vary between Ada run-time systems (RTSs). For identical software running on a variety of equipment under different RTSs, as might be found in the future STARS SEE, it is quite possible that the executions may not be equivalent. A single tool that could analyze the commonality of execution between source-identical systems with Ada multitasking across multiple RTSs would be very useful.

4. REQUIREMENTS

4.1 General Requirements

In general, the analysis system had three requirements: to operate at an Ada source level, to provide some degree of control over the tasks being analyzed, and to be portable across Ada run-time systems.

- Source Level Operation - Operating at the Ada source level will maximize the commonality between Ada and the analyzer. Using Ada constructs to invoke the functionality of the analyzer will provide the user with an understandable, high-level interface.
- Tasking Control - Controlling the execution behavior of tasks will allow the user to exercise the system containing the tasks through various scenarios of execution. Potentially undesirable situations may be uncovered and can then be guarded against.
- Portability - Making the analyzer portable will provide an identical analysis capability among varying Ada runtime systems. Differences in tasking behavior between these systems can be determined and then reconciled within developed software.

4.2 Specific Requirements

The Specific Requirements for the analysis system follow the overall desire for the system's functionality in controlling tasks and providing the user with applicable information. In particular,

1. Vary Task Speeds

a. Call analyzer's Monitor Task from each task.

b. Suspend each task after monitor call.

c. Generate a variable amount of delay on a per task basis.

d. Resume a suspended task.

2. Observe Task Parameter Passing

   a. Pass parameters for entries to Monitor Task.

   b. Examine/Record parameter values as necessary.

3. Selectively Pause, Continue, or Terminate a Task

   a. Similar to Specification 1, but continuation is not delay based.

   b. Action selections are passed to the task for invocation.

4. Gather Task Actvitity Information

   a. Record task instantiations.

   b. Track entry point usage.

5. Provide a Flexible User Interface

5. DEVELOPMENT PLAN

There are two logically and physically distinct parts to the analysis system, a Front-end Preprocessor and the Analyzer Control System. The development of these two parts were separate efforts and are described in the next two sections.

5.1 Front-end Preprocessor

Constructing an analyzer to meet the requirements specified in Section 4 was accomplished by building a partial Ada compiler front-end to preprocess Ada tasking source into an enhanced source. This enhanced source contains additional Ada code to invoke the functionality of the packages imported from the Analyzer Control System. Development of the front-end was top-down and iterative. The plan for developing the front-end proceeded as follows:

1. Token Generation

   a. Strip input source of comments and unnecessary white space,

   b. Build words from individual characters,

   c. Identify words as specific tokens and,

   d. Test token generation.

2. Syntax Analysis

   a. Reduce the grammar to the necessary production set,

   b. Use recursive descent to implement the productions,

c. Build symbol tables for tasking information and,

d. Test syntax analysis.

3. Source Code Instrumentation

a. Identify probe insertion points and insert new code.

b. Test source code instrumentation.

5.2 Analyzer Control System

Development of the Analyzer Control System proceeded in a top-down fashion, from the framework which established a control mechanism, down to the user-selectable control commands. The plan proceeded as follows:

1. Minimal Functionality

a. Construct the Monitor Task as the task control mechanism.

b. Build link tasks from the Monitor Task to the user tasks.

c. Implement a User Interface procedure called by the Monitor Task.

2. Medium Functionality

a. Extend Monitor Task to assign task ID numbers and record task activity.

b. Implement Probe procedures to pass information to the Monitor Task.

c. Enhance User Interface to provide menus, input verification, and user-readable information.

3. Full Functionality

a. Extend Monitor Task to track end of task activity.

b. Modify task entries to carry instrumented tasking information to the Monitor Task.

c. Add Probes to allow modification of execution as commanded by the user (e.g., delay execution by some specified time).

d. Implement a list of conditions to look for.

e. Provide an error handling capability.

f. Complete the User Interface to provide all available information.

6. DESIGN SPECIFICATION

Analyzing tasks at the Ada source level was the major consideration in designing the analysis system. By using Ada statements to interface between the original source and the analyzer, this objective was achieved.

The Analyzer Control System was designed as a group of Ada packages containing the data structures and callable functionality of the analyzer. As Ada packages, the control system can be imported to the program to be analyzed and referenced by additional statements introduced by the front-end. The control system utilizes an intermediary task linkage subsystem that generates

tasks used as links between a master task and the analyzed code. By maintaining rendezvous control over the link tasks, the control system gains control over the analyzed code.

The Front-end Preprocessor takes Ada source code containing tasks as input and generates an augmented Ada source code as output. This augmented code "withs" in the Analyzer Control System packages, sets up links through additional tasks to the original tasks, and provides procedure and entry call statements to the analyzer for information gathering and exchange. The augmented code, when compiled and executed, becomes a custom analyzer for the original code.

The design was carried out in Ada and is reflected as the program specifications of the code in Appendix B.

7. TEST PLAN AND RESULTS

7.1 Front-end Preprocessor

Testing the Front-end Preprocessor is similar to testing a developing compiler implementation. The three phases of the effort, token generation, syntax analysis, and source code instrumentation, provide distinct testing opportunities.

1. Token Generation - the front-end is designed to correctly identify all Ada lexical elements and to generate symbolic tokens for use in the syntax analysis phase. The generator was first tested with small programs containing declarations and tasking structures, then was run through itself to see if a larger program (about 600 statements) with a wide variety of common programming constructs would generate a correct list of tokens. Although not completely testing the entire range of Ada lexical elements, the successful results from this testing were sufficiently convincing to move on to the next phase.

2. Syntax Analysis - the front-end is designed to correctly identify the basic structure of any Ada program. It is not intended to be a complete Ada compiler front-end, so detailed syntax analysis is only performed for certain specific structures regarding declarations and tasking. However, sufficient Ada syntax is incorporated to allow the front-end to correctly accept all legal Ada programs. This was tested by running the Class A & C (compilable and executable) tests from the Ada Compiler Validation Capability (ACVC) test suite (Version 1.10) through the front-end. The ACVC A & C tests (2624 tests) contain virtually all legal Ada structures and combination of structures. The front-end correctly accepted all 2624 tests.

3. Source Code Instrumentation - this final phase of the front-end is designed to incorporate additional Ada statements into the original source input to provide access to the Analyzer Control System. This phase was tested with ACVC tasking tests (199 tests) and with the test programs used in the Analyzer Control System development effort. Although only those test programs from the Analyzer Control System were exercised, they are considered sufficiently representative to declare the front-end as operational.

7.2 Analyzer Control System

Testing the Analyzer Control System was an incremental effort. The first minimally functioning packages, a more robust medium level, and finally the fully functioning system were each tested upon completion.

1. Minimal Implementation - The first stage is to manually insert the instrumenting code into the original source code. At this stage, the monitor system is composed of a minimally functioning Monitor Task and User Interface procedure.
   a. Test Monitor Task
      (1) Accept link logins to show that the link tasks have started.
      (2) Accept probe logins to show that the original tasks have started.
      (3) Accept signals to the monitor from activated probes to show that the probe interface is correct.
   b. Test User Interface
      (1) Display link logins.
      (2) Display probe logins for original tasks.
      (3) Display signals to the monitor from probes reporting that they were activated.
2. Medium Functionality - A data control package is implemented to manage all the information pertinent to the multitasking system. The Monitor Task and the User Interface procedure both manipulate the information in this package.
   a. Test Monitor Task
      (1) Check that data control lists can be properly accessed and that they contain the proper information.
      (2) Verify that probes properly pass in the assigned intertask information.
      (3) Display a message when a particular probe has signaled.
      (4) Confirm that the normal end of program probe is detected and that the monitor has shut down all parts of the monitor system.
   b. Test User Interface
      (1) Exercise the menus for all available operations.
      (2) Check for the proper range of acceptable inputs.
      (3) Verify the correctness of input information.
      (4) Invoke the error routines and test the retry capability.
      (5) Examine the data lists contents for proper and useful information.
3. Full Functionality - the monitor is enhanced to receive and process new information from additional probes and routines. The analyzer allows the user to observe defined conditions and to exert some control over the original tasks.
   a. Test Monitor Task
      (1) Check for nominal task termination.
      (2) Exercise task terminate and abort statements.

(3) Verify that the list containing the module information is updated for terminated tasks, marking the module as dead.

(4) Verify that the list of probes in the terminated task used to update the master probe list are marked as inactive.

b. Test Defined Conditions

(1) Specify the probe to watch for.

(2) Observe the action taken by the monitor.

(3) Check that the probe action is posted.

(4) Verify that the probe procedure executed the indicated action.

c. Test Error Handling Capability

(1) Check that error messages regarding the type of error and where it occurred are displayed.

(2) Exercise exception handlers placed in the monitor task to report primarily tasking related exceptions.

d. Test User Interface

(1) Display parents of task entry calls and the frequency of those calls.

(2) Display the number of calls total on each entry.

(3) Display the maximum length of each entry queue.

8. CONCLUSION

This system provides an initial tool for analyzing and experimenting with multitask Ada software. The User Interface is simplistic and awkward to use, but the primary goal of the prototype was to develop and demonstrate the analysis principles and to design, develop, and implement a completely Ada portable tool. Further work to extend the User Interface and other system capabilities would make the system easier to apply and use. Refinements to improve or extend this system are left to the interested Software Engineer.

A. USER'S GUIDE

This guide is intended for use by those already familiar with the concepts related to conventional debuggers. The facilities provided by this system are tailored to monitor and manage Ada intertask relationships. When properly applied, this system should provide the user with a useful tool for analyzing intertask problems as well as a tool for extracting potentially valuable run-time information regarding the interactions of the tasks present in the software system.

There are three parts to the analysis process. First, the source to be analyzed is instrumented with Ada code providing connections to the Ada packages comprising the analyzer. These connections or *probes* allow the source to have access to the analyzer's functionality. Next, this enhanced source is compiled and linked to produce an executable image. Finally, the program is run as normal. However, when the first instrumenting probe is encountered, the analyzer's main menu will be displayed. Now the original source program is being monitored and controlled via the analyzer.

Section A.1 describes the details surrounding the process of instrumenting the source code, such as what library packages are required, where the inserted code is placed, and how to set the parameters of the code. Section A.2 discusses the internal data structures and the information they contain and Section A.3 shows how to manipulate this information. Finally, Section A.4 gives a complete example of using the automatic front-end preprocessor to instrument a source program.

A.1 Instrumenting the Source

The process of instrumenting the source is relatively straightforward and is carried out by the analyzer's front-end preprocessor. First the groups, or logical modules, are determined. These are task bodies, generic packages, and if included, the main program. The source code probes are placed as local declarations in the modules so that whenever a new instantiation or elaboration occurs, a new set of probes is generated. The names of the probes within these sets will be identical therefore each instantiation must be assigned a unique module ID which the probes and the monitor use to keep the identical names straight.

The instrumenting code is composed of several parts. There is a link initialization package which instantiates a new link task and performs a login to the monitor task where a unique module ID is assigned. Then a probe is activated which allows the monitor to pause the task and prevent further startup, if desired by the user. A generic procedure is instantiated for each probe with the parameters used to initialize internal data structures. Procedure calls are placed at appropriate locations in the source code to invoke the instantiated generic procedures.

A packaging scheme is used whereby the link initialization package, the link task it sets up, and the probes that will use the link are all placed in a package. There is one such package per group.

A.1.1 Library Packages

With this release of the analyzer, there are five library packages that the front-end "withs" in the source code.

1. MTD_FUNDAMENTAL_TYPES - contains the base or fundamental types used to build the more complex data types.

2. MTD_COMPLEX_TYPES - contains the complex record data types used by the software.
3. USEFUL_TYPES - contains several miscellaneous types.
4. MTD_TOOL - contains both the generic procedure definition for the probes but also the monitor task as well.
5. LINK_INIT - is a generic package that generates the link task, several data structures, and the initializing code to log the link in to the monitor, receive the module ID, and issue a task initialization probe to the monitor.

A.1.2 Packaging the Links and Probes

A package is defined to hold the instantiation of the generic link initialization package and the instantiations of the generic probe procedures for each group.

A.1.2.1 Determining the Groups

Groups are considered templates for modules that can be created both at compile time and at run-time in a dynamic manner. They are :

- Generic packages,
- Task bodies and,
- Source files, to catch all task interaction points not included in the preceeding groups.

Each instantiation of a generic package or task type will generate a new module from the base group template (the generic or task type definition). If the task is not used as a type, then the group will have only one module. Otherwise, each group will have potentially many instantiations with many modules. Each module is assigned a unique module ID so that the module and its probes can be properly managed by the monitor task.

A.1.2.2 Generic Link Initialization Package

The generic link initialization package includes the data structures needed to describe the module it is in, create a new link task, and log in to the monitor task, receiving a unique ID in return. Overloading and scoping is used to allow the use of the same names for the data variables and link task name. An instantiation of the link initialization package is as follows:

```
package mtd_inserted_instruments_yyy is

  package init_link is new link_init(
    m_group=>
    f_name=>
    m_name=>

              m_type=>
    m_modifier=>is_normal
  ); use init_link;

end mtd_inserted_instruments_yyy;
```

The generic parameters are set to values identifying the link and the group it resides in.

The variables are as follows:

- m_group - the group number assigned at instrumentation time. Upon execution, a unique integer is assigned by the monitor to form a group-instantiation combination, referred to as the module ID.
- f_name - is the file name containing the group.
- m_name - the path from the root (top of file) down through any packages, procedures, and tasks necessary to follow to reach the module. A record access format is used to specify each level.
- m_type - is the type of module (i.e., task, package, or file).
- m_modifier - description modifier to tag the package as normal, generic, or type.

A.1.2.3 Generic Probe Procedures

The instantiations of the probe procedures are the same for each probe in the group, with the differences only in the names and the task action the probe monitors. There are two additional variables needed for each probe procedure. The first is a boolean flag to mark the first time the procedure is called. A unique ID number is returned from this initial call in the second variable.

An instantiation of a probe procedure would look like the following:

```
probe_xxx_first_time:BOOLEAN:=TRUE;
probe_xxx_id:P_ID:=-1;
procedure probe_xxx is new probe(
  p_name=>"probe_xxx",
  t_action=>
  p_path=>
  );
```

Where the xxx is replaced with an integer number. The generic variable t_action is set to the type of tasking action being monitored and the p_path variable is set to a string representing the path that must be followed to reach the procedure call. This is measured from the location of the group definition down through packages, procedures, etc.

The generated name of the probe procedure is not important as long as it is consistent with the calls and unique within the group. Scoping will take care of duplicate names in different groups.

A.1.3 Task Entry Parameter Modifications

In order to track parent-child execution paths, two parameters must be added to task entry definitions in the original source. They are the parent's module ID and a string with the actual name used to call the task. This is necessary when dealing with task types since the type can be assigned to any valid variable name and there is no Ada mechanism to convert the task's current name into a string for manipulation. These two additional pieces of information are passed into the monitor by the probes following the task accept statements.

For example, the original specification for a task entry might be the following:

```
entry one(
  p1:INTEGER;
  p2:INTEGER;
  p3:STRING );
```

while the modified entry would be:

```
entry one(
  module_identity:MODULE_ID;
  task_called_as:STRING;
  p1:INTEGER;
  p2:INTEGER;
  p3:STRING );
```

where the type MODULE_ID is defined in the type package mtd_complex_types.

A.1.4 Inserting the Probes

A probe inserted into the original code is actually a procedure call to a previously instantiated generic procedure. The procedure parameters are used to allow Ada scoping rules to pick the currently visible definition of a module variable and to pass in current analyzer values. These calls are placed before and after task interaction statements in the source. In some cases it is not possible to place a statement before the action but it is possible to place it after the action, such as an accept statement within a select block. The enumeration types associated with the tasking action being monitored reflect these possibilities. This enumeration type is defined in the package mtd_fundamental_types.

A.1.4.1 Determining Locations

Probes are placed before the following types of Ada statements, if possible:

- Task calls,
- Select blocks,
- Task aborts,
- Raise exceptions,
- Delays,
- Accepts,
- Start of the rendezvous code,
- End of rendezvous code,
- Task ends and,
- The end of the program.

In addition, probes may be placed after the following:

- Delays,

- Task calls and,
- Select blocks.

Certain language constructs will prevent some of the placements listed above from being followed, such as conditional accept blocks using selects or conditional task entry calls using select blocks. When this happens, a probe after the event is placed in the source to at least notify the monitor that a particular type of event has just occurred.

A.1.4.2 The Parameters

An inserted probe procedure call has the following form:

```
probe_xxx(
  link_task=>init_link.module_dynamic_info.link_task,
     module=>init_link.module_dynamic_info.id,
     id=>probe_xxx_id,
     first_time=>probe_xxx_first_time
     );
```

In the above, the xxx is replaced with the integer number corresponding to the name used for the instantiation. Scoping rules allow the use of a standard name for the link task pointer and the present module ID.

There are three additional parameters that have default values. These are:

- parent_module of type MODULE_ID found in mtd_complex_types
- task_call_name of type STRING_REC found in useful_types
- number_queued of type NATURAL

The only parameter with a somewhat ambiguous name is the last parameter which is used to pass the current length of the queue associated with the accept the probe is monitoring. The values used for the first two are passed in as added parameters on the accept statement. Only probes following accept statements use these parameters and the defaults allow omitting them for most of the probe procedure calls.

A.1.5 Compiling the Source

Once the instrumentation process has been completed it is only necessary to compile the source files and link them into an executable image. When run, the first probe procedure to login will cause the analyzer's main menu to come up on the screen.

A.2 Internal Data Lists and Their Information

All information regarding the executing program is passed to the monitor task via link and probe logins and probe signals to the monitor. This data is placed into lists which are managed by a combination of a data control package, procedures optimized to handle the different types of information, and a generic list manager.

In order to get the most out of the analysis system, the user must understand the information that the system is managing and what it means. The remainder of this section presents the separate lists of data, describes the contents of the lists, and indicates the relationships between them, if any exists.

A.2.1 The Group List

This list contains information about each group that has logged into the monitor task. The information is the static data common to all modules that are or will be included in the group. A display of the list of groups is as follows:

Group Details:

Group ID: 2
Module File: main_test.a
Module Name: main_test.alpha
Module Type: IS_TASK
Module Modifier: IS_NORMAL
Number of instantiations: 1

The Module File is the name of the Ada source file containing the group. Module Name is the name of the generic package, task, or file, depending upon the Module Type. The Module Modifier tells whether the module is a generic, task type, or neither of these. The number of instantiations tracks the instantaneous number of modules derived from the group.

A.2.2 The List of Modules for a Group

The module list for a group contains information on an instantiation of the group template. This list could be very dynamic when dealing with task types that are dynamically created and terminated by the program. The information contained in this list for one such module is:

Module Details:

Present Module ID: 2 , 1
Link Action: PASS_INFO
Child is executing?: FALSE at Probe: -1

The Link Action field can hold three states:

- Pass information,
- Pass information and terminate link task or,
- Terminate link task.

Additonal information provided is a record of whether a task entry has been called from this module. If it has, then this module is waiting for the completion of the rendezvous code for that entry. The probe ID is the probe in the task's rendezvous code block.

A.2.3 The Probe List

The master list of all probes currently logged in tracks both the static and dynamic information regarding the probes. One such entry would look like this:

Probe Details:

Probe ID: 5
Probe Name: probe_004
Task Action: START_RENDEZVOUS
Probe Path: .
Probe Action: NOP
Select Exception: 1
Delay Value: 1.000
Probe Waiting: TRUE
Parent Module ID: 1 , 1
Task Called As: alpha.one
Present Module ID: 2 , 1

The first five fields are as described previously. The remainder are as follows:

- Select Exception - indicates the exception selected.
- Delay Value - used by the probe when it is instructed to delay execution.
- Probe Waiting - indicates if the probe has the task paused.
- Parent module ID - is the module ID of the parent module that has made a task entry call. This is only applicable to the start rendezvous block probes.
- Task Called As - is the name the task has been assigned to if it is a task type or the name of the task if not.
- Present Module ID - is the module ID the probe exists in.

A.2.4 The List of Probes for a Module

Each module instantiation has a list of those probes that are contained within it. When displayed the following is shown.

List of probes for module: 2 , 1

Probe ID: 1 probe_init
Probe ID: 2 probe_002
Probe ID: 5 probe_004

The probe's ID assigned by the monitor is listed along with the name that was assigned at instrumentation time.

A.2.5 The Probe Condition List

The probe condition list is used to specify conditions which cause the monitor to take a predefined action. This action may be to report to the user that a specified probe(s) has been executed, prevent further execution of that module until the user releases the probe, or modify the execution of the module in one of the several ways discussed earlier.

The information required to specify these three conditions are slightly different and this shows up when the three condition lists are displayed.

- Reporting activation of a probe.

  List of probes assigned for reporting:

  Report Condition: 1

  Module: 1, 1
  Name: probe_001
  Probe ID: 4
  Task Action: TASK_CALL_BEFORE
  Condition action: REPORT
  Check what?: CHECK_TASK_ACTION

- Breakpointing the execution of the module.

  List of probes for breakpointing:

  Break Condition 2

  Module: 2, 1
  Name: probe_004
  Probe ID: 5
  Task Action: START_RENDEZVOUS
  Condition action: BREAK
  Check what?: CHECK_TASK_ACTION
  Break point status: TRIPPED

- Controlling the execution of the module.

  List of probes for execution control:

  Control Condition 3

  Module: -1, -1
  Name:
  Probe ID: -1
  Task Action: END_OF_PROGRAM
  Condition action: CONTROL
  Check what?: CHECK_TASK_ACTION
  Probe Control Action: DELAY_TASK
  Delay value: 1.000

A.3 Monitoring and Managing Task Interrelationships

This section discusses how to use the analyzer data and how to manipulate the probes and the actions they can perform.

Through the proper use of these facilities it is possible to modify execution flow of the program, be notified when an indicated event occurs, monitor task entry statistics, and actually control when and the order in which task rendezvous can occur.

A.3.1 Specifying a Probe to be Monitored

Probes may be assigned to one of three categories for inclusion on the condition list. They are:

- Reporting
- Breakpointing
- Execution modification and control

Once the category is selected, the probe to be included must be specified in one of 4 ways, or EXIT out, as the following menu illustrates.

1. Specify by probe ID number
2. Specify by probe name
3. Specify by module ID
4. Specify by task action
5. EXIT

After the probe is selected, the user may be prompted for more information, depending upon which category the probe is to be placed. If execution control is desired, the user must select the type of action the probe is to perform when the monitor releases it and any auxiliary information to carry out the operation, such as a delay value to use.

A.3.2 Action to be Taken When a Specified Probe is Detected

When the probe designated is activated, the user interface will display detailed information about the probe, as is shown here.

Display when probe matches an entry on condition list:
Probe Details:

Probe ID: 4
Probe Name: probe_001
Task Action: TASK_CALL
Probe Position: BEFORE
Probe Action: NOP
Select Exception: 1
Delay Value: 1.00000
Probe Waiting: TRUE
Parent Module ID: -1 , -1
Task Called As: alpha.one
Present Module ID: 1 , 1

A.3.2.1 Reporting

Probe reporting occurs when execution reaches a selected point or region of code, or type of probe. This activates the user interface and allows the user to examine all data in the monitor's data base and to set new conditions on the condition list. When the user interface is exited, the program will automatically continue its normal execution.

A.3.2.2 Breakpoints

A breakpoint operates as a reporting probe except that it must be released by a command from the user. It is *not* automatically released. This is useful when the user wishes to hold up the execution of one module until other modules are in a desired execution state or point in the code.

A.3.2.3 Modify Execution

When a probe is specified to control execution it is assigned an operation to carry out upon release by the monitor along with qualifying information. The probe is not reported to the user but when it is released (all probes cause a pause in execution of the enclosing module) the operation to perform and the auxiliary information is passed back to it for processing.

For example, you can select all probes in a module to cause a delay of .01 second when activated. Or you could set one probe located in a module at a rendezvous block to raise a tasking exception when activated. The manner in which probes for this operation are specified is very flexible.

A.4 Automatic Instrumentation

This section gives a demonstration of the automatic instrumentation process. It is important to note here that the front-end needs to have a marker to identify the main unit in the source input. The user must add a "pragma task_analyzer" statement at the appropriate spot in the code. Also, the front-end assumes that the source input is a legal, compilable Ada program. Submitting illegal source to the front-end will cause it to abort.

A.4.1 Using the Front-end Preprocessor

The following shows the virtually turn-key usage of the front-end. The user is prompted for a source file and is given the option to enter the name for the destination file. Queries from the front-end are highlighted in boldface.

Enter source file name : orig_main_test.a

Destination file name (default : new_orig_main_test.a) :

A.4.2 Input - Original Ada Program

The user supplied "pragma task_analyzer" statement is highlighted in boldface.

```
with text_io;use text_io;

package main_test is
task alpha is

   entry one(count:INTEGER);

   entry two(count:INTEGER);
end alpha;
```

```
end main_test;

package body main_test is
task body alpha is
package int_io is new integer_io(INTEGER);use int_io;
keep:INTEGER;

begin

loop

  accept one(count:INTEGER) do

    put("Task ONE, ");
    put("Count=");put(count);
    new_line;

    keep:=count;
  end one;

  accept two(count:INTEGER) do

    put("Task TWO, Count=");
    put(count);new_line;
  end two;

  exit when keep=10;
end loop;
end ;

end main_test;

pragma task_analyzer;

with main_test;use main_test;
with text_io;use text_io;

procedure main is


begin
  put_line("Main test procedure has begun.");

  for i in 1..10 loop
  alpha.one(i);

  alpha.two(i);
```

```
   end loop;
end main;
```

A.4.3 Output - Instrumented Program

Source code added by the front-end is highlighted below in boldface.

```
with mtd_fundamental_types; use mtd_fundamental_types;
with useful_types; use useful_types;
with mtd_complex_types; use mtd_complex_types;
with mtd_tool; use mtd_tool;
with link_init;
with new_a_strings; use new_a_strings;

with text_io;use text_io;

package main_test is
task alpha is

   entry one(count:INTEGER;
module_identity : module_id; task_called_as : string);

   entry two(count:INTEGER;
module_identity : module_id; task_called_as : string);
end alpha;

end main_test;

with mtd_fundamental_types; use mtd_fundamental_types;
with useful_types; use useful_types;
with mtd_complex_types; use mtd_complex_types;
with mtd_tool; use mtd_tool;
with link_init;
with new_a_strings; use new_a_strings;

package body main_test is
task body alpha is

package instruments_2 is
  package init_link_2 is new link_init(
    m_group => 2,
    f_name => "orig_main_test.a",
    m_name => ".",
    m_type => IS_TASK,
    m_modifier => IS_NORMAL
    );
```

```
use init_link_2;

probe_1_first_time : boolean := true;
probe_1_id : p_id := -1;
procedure probe_1 is new probe(
  p_name => "probe_1",
  t_action => START_ACCEPT_BEFORE,
  p_path => "."
  );

probe_2_first_time : boolean := true;
probe_2_id : p_id := -1;
procedure probe_2 is new probe(
  p_name => "probe_2",
  t_action => START_RENDEZVOUS,
  p_path => "."
  );

probe_3_first_time : boolean := true;
probe_3_id : p_id := -1;
procedure probe_3 is new probe(
  p_name => "probe_3",
  t_action => END_ACCEPT,
  p_path => "."
  );

probe_4_first_time : boolean := true;
probe_4_id : p_id := -1;
procedure probe_4 is new probe(
  p_name => "probe_4",
  t_action => START_ACCEPT_BEFORE,
  p_path => "."
  );

probe_5_first_time : boolean := true;
probe_5_id : p_id := -1;
procedure probe_5 is new probe(
  p_name => "probe_5",
  t_action => START_RENDEZVOUS,
  p_path => "."
  );

probe_6_first_time : boolean := true;
probe_6_id : p_id := -1;
procedure probe_6 is new probe(
  p_name => "probe_6",
  t_action => END_ACCEPT,
```

```
    p_path => "."
    );

  probe_7_first_time : boolean := true;
  probe_7_id : p_id := -1;
  procedure probe_7 is new probe(
    p_name => "probe_7",
    t_action => TASK_END,
    p_path => "."
    );
end instruments_2;
use instruments_2;

package int_io is new integer_io(INTEGER);use int_io;
keep:INTEGER;

begin
loop

    probe_1(
      link_task => init_link_2.module_dynamic_info.link_task,
      module => init_link_2.module_dynamic_info.id,
      id => probe_1_id,
      first_time => probe_1_first_time
      );

   accept one(count:INTEGER;
module_identity : module_id; task_called_as : string) do

    probe_2(
      link_task => init_link_2.module_dynamic_info.link_task,
      module => init_link_2.module_dynamic_info.id,
      id => probe_2_id,
      first_time => probe_2_first_time,
      parent_module => module_identity,
      task_call_name => to_a(task_called_as),
      number_queued => ALPHA.ONE'COUNT
      );

     put("Task ONE, ");
     put("Count=");put(count);
     new_line;
     keep:=count;

    probe_3(
      link_task => init_link_2.module_dynamic_info.link_task,
      module => init_link_2.module_dynamic_info.id,
```

```
      id => probe_3_id,
      first_time => probe_3_first_time,
      parent_module => module_identity,
      task_call_name => to_a(task_called_as)
      );

  end one;

   probe_4(
     link_task => init_link_2.module_dynamic_info.link_task,
     module => init_link_2.module_dynamic_info.id,
     id => probe_4_id,
     first_time => probe_4_first_time
     );

  accept two(count:INTEGER;
module_identity : module_id; task_called_as : string) do

   probe_5(
     link_task => init_link_2.module_dynamic_info.link_task,
     module => init_link_2.module_dynamic_info.id,
     id => probe_5_id,
     first_time => probe_5_first_time,
     parent_module => module_identity,
     task_call_name => to_a(task_called_as),
     number_queued => ALPHA.TWO'COUNT
     );

    put("Task TWO, Count=");
    put(count);new_line;

   probe_6(
     link_task => init_link_2.module_dynamic_info.link_task,
     module => init_link_2.module_dynamic_info.id,
     id => probe_6_id,
     first_time => probe_6_first_time,
     parent_module => module_identity,
     task_call_name => to_a(task_called_as)
     );

  end two;
  exit when keep=10;
end loop;

   probe_7(
     link_task => init_link_2.module_dynamic_info.link_task,
     module => init_link_2.module_dynamic_info.id,
```

```
      id => probe_7_id,
      first_time => probe_7_first_time
      );

end ;

end main_test;

with mtd_fundamental_types; use mtd_fundamental_types;
with useful_types; use useful_types;
with mtd_complex_types; use mtd_complex_types;
with mtd_tool; use mtd_tool;
with link_init;
with new_a_strings; use new_a_strings;

pragma task_analyzer;

with main_test;use main_test;
with text_io;use text_io;

procedure main is

package instruments_3 is
  package init_link_3 is new link_init(
    m_group => 3,
    f_name => "orig_main_test.a",
    m_name => ".",
    m_type => IS_PROCEDURE,
    m_modifier => IS_NORMAL
    );
  use init_link_3;

  probe_8_first_time : boolean := true;
  probe_8_id : p_id := -1;
  procedure probe_8 is new probe(
    p_name => "probe_8",
    t_action => TASK_CALL_BEFORE,
    p_path => "."
    );

  probe_9_first_time : boolean := true;
  probe_9_id : p_id := -1;
  procedure probe_9 is new probe(
    p_name => "probe_9",
    t_action => TASK_CALL_BEFORE,
    p_path => "."
    );
```

```
  probe_10_first_time : boolean := true;
  probe_10_id : p_id := -1;
  procedure probe_10 is new probe(
    p_name => "probe_10",
    t_action => END_OF_PROGRAM,
    p_path => "."
    );
end instruments_3;
use instruments_3;

begin
   put_line("Main test procedure has begun.");

   for i in 1..10 loop

    probe_8(
      link_task => init_link_3.module_dynamic_info.link_task,
      module => init_link_3.module_dynamic_info.id,
      id => probe_8_id,
      first_time => probe_8_first_time,
      task_call_name => to_a("ALPHA.ONE")
      );

   alpha.one(i,
init_link.module_dynamic_info.id, "ALPHA.ONE");

    probe_9(
      link_task => init_link_3.module_dynamic_info.link_task,
      module => init_link_3.module_dynamic_info.id,
      id => probe_9_id,
      first_time => probe_9_first_time,
      task_call_name => to_a("ALPHA.TWO")
      );

   alpha.two(i,
init_link.module_dynamic_info.id, "ALPHA.TWO");

   end loop;

    probe_10(
      link_task => init_link_3.module_dynamic_info.link_task,
      module => init_link_3.module_dynamic_info.id,
      id => probe_10_id,
      first_time => probe_10_first_time
      );

end main;
```

B. PROGRAM LISTING

```
--
--
--        DISCLAIMER OF WARRANTY AND LIABILITY
--
--
-- THIS IS EXPERIMENTAL PROTOTYPE SOFTWARE.  IT IS PROVIDED "AS IS"
-- WITHOUT WARRANTY OR REPRESENTATION OF ANY KIND.  THE INSTITUTE
-- FOR DEFENSE ANALYSES (IDA) DOES NOT WARRANT, GUARANTEE, OR MAKE
-- ANY REPRESENTATIONS REGARDING THIS SOFTWARE WITH RESPECT TO
-- CORRECTNESS, ACCURACY, RELIABILITY, MERCHANTABILITY, FITNESS FOR
-- A PARTICULAR PURPOSE, OR OTHERWISE.
--
-- USERS ASSUME ALL RISKS IN USING THIS SOFTWARE.  NEITHER IDA NOR
-- ANYONE ELSE INVOLVED IN THE CREATION, PRODUCTION, OR DISTRIBUTION
-- OF THIS SOFTWARE SHALL BE LIABLE FOR ANY DAMAGE, INJURY, OR LOSS
-- RESULTING FROM ITS USE, WHETHER SUCH DAMAGE, INJURY, OR LOSS IS
-- CHARACTERIZED AS DIRECT, INDIRECT, CONSEQUENTIAL, INCIDENTAL,
-- SPECIAL, OR OTHERWISE.
--
--
```

B.1 Front-end Preprocessor

```
--
-- ******************************************
-- Portable Ada Multitasking Analyzer System
--
--          Version 1.0
--
--  Designed, developed, and written by:
--          Robert J. Knapper
--          David O. LeVan
--             of the
-- Computer Software and Engineering Division
--     Institute for Defense Analyses
--          Alexandria, VA
--
--             11/8/88
--
-- ******************************************
--
-- The following is a recognizer for the syntax of the Ada
-- language. This recognizer has been developed to be used
-- as part of an instrumentation tool for the IDA/STARS
-- Portable Ada Multitasking Analysis System (PAMAS).
--
-- Since the input to PAMAS is intended to be syntactically
-- legal Ada, this recognizer makes no attempt at error
-- detection or correction.
--
-- The recognizer has two parts. A lexical scanner that produces
-- a stream of tokens from the input file, and a "mini" parser to
-- recognize tasking structures. The recognizer uses a limited
-- symbol table to hold infc-mation concerning task types, access
-- types, and any appropriate objects. With the addition of error
-- detection/correction, a complete symbol, and full parser, this
-- recognizer would be a viable front-end for an Ada compiler.
--
-- Institute for Defense Analyses
-- 1801 N. Beauregard Street
-- Alexandria, VA 22311

with text_io; use text_io;
procedure pamas_front_end is

  subtype line is string;
  max_length : constant natural := 256;
```

```
  read_file, stripped_file, write_file, instr_file, test_file,
   merge1_file, merge2_file, merged_file : file_type;
  test_file_name, main_file_name : line(1..max_length) := (others => ' ');
  test_name_length, main_name_length, last : natural := 0;

  subtype format_effectors is character range  ASCII.NUL..ASCII.US;
  answer : string(1..2) := "N ";

  applicable : boolean := true;

-- The "strip_comments" procedure removes all Ada comments from the
-- input file. This facilitates the scanning of the input file for
-- instrumentation of the code with AMAS structures.

  procedure strip_comments is
    line_buffer, out_buffer : line(1..max_length) := (others => ' ');
    line_length, line_position, out_position : natural := 0;
    last_real_character_position : natural := 0;
    comment_detected, string_detected : boolean := false;
  begin
    while not end_of_file (read_file) loop
      get_line (read_file, line_buffer, line_length);


      while line_position < line_length loop
        line_position := line_position + 1;
        if line_buffer(line_position) = '"' then
          string_detected := not string_detected;
        end if;
        if line_buffer(line_position) = '-' and
           line_buffer(line_position+1) = '-' and
                      not string_detected then
          if last_real_character_position > 0 then
            put_line (stripped_file,
              out_buffer(1..last_real_character_position));
        end if;
        exit;
      else
        out_position := out_position + 1;

-- replace all format effector characters by blanks

            if line_buffer(line_position) in format_effectors then
                 out_buffer(out_position) :=' ';
            else
           out_buffer(out_position) := line_buffer(line_position);
            end if;
```

```
          if line_position = line_length then
            while out_buffer(line_length) = ' ' loop
              line_length := line_length - 1;
              exit when line_length = 0;
            end loop;
            if line_length > 0 then
              put_line (stripped_file, out_buffer(1..line_length));
            end if;
          end if;
        end if;
        if line_buffer(line_position) /= ' ' then
          last_real_character_position := line_position;
        end if;
      end loop;
      line_position := 0;
      out_position := 0;
      last_real_character_position := 0;
    end loop;
  end strip_comments;

-- The "setup" procedure reads in the name of the file to be scanned
-- and creates an output file for the scan results. If no name is
-- given for the destination file, a default is generated.

  procedure set_up is
    source_name, destination_name,
      default_name : line(1..max_length) := (others => ' ');
    name_length, default_length : natural := 0;
  begin
    new_line;
    put ("Enter source file name : ");
    get_line (source_name, name_length);
    main_file_name := source_name;
    main_name_length := name_length;
    default_length := name_length + 4;
    default_name(1..default_length) :=
                    "new_" & source_name(1..name_length);
    open (read_file, in_file, source_name(1..name_length));

    new_line;
    put ("Destination file name (default : " & "new_" &
              source_name(1..name_length) & ") : ");
    get_line (destination_name, name_length);
    if name_length = 0 then
      create (merged_file, out_file, default_name(1..default_length));
    else
      create (merged_file, out_file, destination_name(1..name_length));
```

```
    end if;
    create (stripped_file, out_file, "xtemp.strip");
    strip_comments;
    close (stripped_file);
    close (read_file);
    open (read_file, in_file, "xtemp.strip");
    create (instr_file, out_file, "instr.file");
    create (write_file, out_file, "xtemp.output");
  end set_up;

  procedure test_harnass is
    source_name, destination_name,
      default_name : line(1..max_length) := (others => ' ');
    name_length, default_length : natural := 0;
  begin
    applicable := true;
    get_line(test_file, source_name, name_length);
    main_file_name := source_name;
    main_name_length := name_length;
    if source_name(1) = '*' then
      applicable := false;
      return;
    end if;
    default_length := name_length + 4;
    default_name(1..default_length) :=
                    "new_" & source_name(1..name_length);
    open (read_file, in_file, source_name(1..name_length));
    new_line;
    put_line ("Destination file is " & default_name(1..default_length));
    new_line;
    create (merged_file, out_file, default_name(1..default_length));
    create (write_file, out_file, "xtemp.output");
    create (stripped_file, out_file, "xtemp.strip");
    strip_comments;
    close (stripped_file);
    close (read_file);
    open (read_file, in_file, "xtemp.strip");
  end test_harnass;

  procedure scan is separate;

  procedure merge is separate;

begin -- main


new_line(2);
```

```
put_line("        Portable Ada Multitasking Analyzer System");
put_line("              Version 1.0");
new_line;
put_line("--");
put_line("--");
put_line("--       DISCLAIMER OF WARRANTY AND LIABILITY");
put_line("--");
put_line("--");
put_line("--  THIS IS EXPERIMENTAL PROTOTYPE SOFTWARE.  IT IS PROVIDED ""AS IS"" ");
put_line("--  WITHOUT WARRANTY OR REPRESENTATION OF ANY KIND.  THE INSTITUTE");
put_line("--  FOR DEFENSE ANALYSES (IDA) DOES NOT WARRANT, GUARANTEE, OR MAKE");
put_line("--  ANY REPRESENTATIONS REGARDING THIS SOFTWARE WITH RESPECT TO");
put_line("--  CORRECTNESS, ACCURACY, RELIABILITY, MERCHANTABILITY, FITNESS FOR");
put_line("--  A PARTICULAR PURPOSE, OR OTHERWISE.");
put_line("--");
put_line("--  USERS ASSUME ALL RISKS IN USING THIS SOFTWARE.  NEITHER IDA NOR");
put_line("--  ANYONE ELSE INVOLVED IN THE CREATION, PRODUCTION, OR DISTRIBUTION");
put_line("--  OF THIS SOFTWARE SHALL BE LIABLE FOR ANY DAMAGE, INJURY, OR LOSS");
put_line("--  RESULTING FROM ITS USE, WHETHER SUCH DAMAGE, INJURY, OR LOSS IS");
put_line("--  CHARACTERIZED AS DIRECT, INDIRECT, CONSEQUENTIAL, INCIDENTAL,");
put_line("--  SPECIAL, OR OTHERWISE.");
put_line("--");
put_line("--");

put("Press RETURN key to continue:");get_line(answer,last);

  new_line;
  put_line("Test harness mode requires submission of a file containing file names to process.");
  put_line("This is for validation purposes only.");
  new_line;
 put ("Run in test harnass mode? (y/n) : ");
 get_line (answer, last);
 if answer(1) = 'y' or answer(1) = 'Y' then
   new_line;
   put ("Enter file name for tests : ");
   get_line (test_file_name, test_name_length);
   open (test_file, in_file, test_file_name(1..test_name_length));
   while not end_of_file (test_file) loop
     test_harnass;
     if applicable then
       scan;
       close (read_file);
       close (write_file);
     end if;
   end loop;
 else
   set_up;
```

```
    scan;
    close (write_file);
    close (instr_file);
    open (merge1_file, in_file, "xtemp.output");
    open (merge2_file, in_file, "instr.file");
    merge;
    close (merged_file);
  end if;

exception
  when constraint_error => put_line("CONSTRAINT_ERROR");
  when program_error => put_line("PROGRAM_ERROR");
  when storage_error => put_line("STORAGE_ERROR");
  when others =>
          put_line("Exception raised");
          close (write_file);
end pamas_front_end;
```

```
--
-- ******************************************
-- Portable Ada Multitasking Analyzer System
--
--          Version 1.0
--
--  Designed, developed, and written by:
--          Robert J. Knapper
--          David O. LeVan
--             of the
-- Computer Software and Engineering Division
--     Institute for Defense Analyses
--          Alexandria, VA
--
--             11/8/88
--
-- ******************************************
--
-- The "scan" procedure performs a lexical scan of the input file,
-- generating a stream of tokens (lexical elements). These tokens
-- are used (with appropriate symbol tables) as input to the mini
-- tasking parser.

separate (pamas_front_end)
procedure scan is

  type token_type is (identifier,

  -- Statement beginning tokens

                      aborttoken, accepttoken, begintoken, casetoken,
                      declaretoken, delaytoken, exittoken, fortoken,
                      gototoken, iftoken, looptoken, nulltoken,
                      raisetoken, returntoken, selecttoken, whiletoken,

  -- All other reserved words

                      abstoken, accesstoken, alltoken, andtoken,
                      arraytoken, attoken, bodytoken, constanttoken,
                      deltatoken, digitstoken, dotoken, elsetoken,
                      elsiftoken, endtoken, entrytoken, exceptiontoken,
                      functiontoken, generictoken, intoken, istoken,
                      limitedtoken, modtoken, newtoken, nottoken,
                      oftoken, ortoken, otherstoken, outtoken,
                      packagetoken, pragmatoken, privatetoken,
                      proceduretoken, rangetoken, recordtoken, remtoken,
                      renamestoken, reversetoken, separatetoken,
```

```
            subtypetoken, tasktoken, terminatetoken,
            thentoken, typetoken, usetoken, whentoken,
            withtoken, xortoken,

-- Special symbols

            sharp, quotation, ampersand, tick,
            leftparen, rightparen, semicolon, comma,
            plus, minus, divide, star, less, equal,
            greater, dot, colon, verticalbar, exclamation,
            dollar, percent, question, atsign, leftsquare,
            rightsquare, backslash, circumflex, grave,
            leftbrace, rightbrace, tilde, underscore,
            arrow, doubledot, exponentiate, becomes,
            notequal, greaterequal, lessequal, leftlabel,
            rightlabel, box,

-- Miscellaneous tokens

            charliteral, stringliteral, numliteral,
            comment, other);

subtype reserved is token_type range aborttoken .. xortoken;
subtype statement_tokens is token_type range identifier .. whiletoken;
subtype simple_operators is token_type range plus..greater;
subtype delimiter is token_type range ampersand .. verticalbar;
subtype simple_delimiter is token_type range minus .. colon;
subtype compound_delimiter is token_type range arrow .. box;
subtype line_enders is token_type range quotation .. verticalbar;

subtype graphic_chars is character range ' '..'~';
subtype format_effectors is character range  ASCII.NUL..ASCII.US;
subtype letter is character range 'A'..'Z';
subtype digit is character range '0'..'9';


subtype word_type is string(1..10);

package token_io is new enumeration_io(token_type);

reserved_word_list : array(reserved) of word_type :=
            ("ABORT    ", "ACCEPT   ", "BEGIN    ",
             "CASE     ", "DECLARE  ", "DELAY    ",
             "EXIT     ", "FOR      ", "GOTO     ",
             "IF       ", "LOOP     ", "NULL     ",
             "RAISE    ", "RETURN   ", "SELECT   ",
             "WHILE    ", "ABS      ", "ACCESS   ",
```

```
                "ALL     ", "AND     ", "ARRAY   ",
                "AT      ", "BODY    ", "CONSTANT ",
                "DELTA   ", "DIGITS  ", "DO      ",
                "ELSE    ", "ELSIF   ", "END     ",
                "ENTRY   ", "EXCEPTION ", "FUNCTION ",
                "GENERIC ", "IN      ", "IS      ",
                "LIMITED ", "MOD     ", "NEW     ",
                "NOT     ", "OF      ", "OR      ",
                "OTHERS  ", "OUT     ", "PACKAGE ",
                "PRAGMA  ", "PRIVATE ", "PROCEDURE ",
                "RANGE   ", "RECORD  ", "REM     ",
                "RENAMES ", "REVERSE ", "SEPARATE ",
                "SUBTYPE ", "TASK    ", "TERMINATE ",
                "THEN    ", "TYPE    ", "USE     ",
                "WHEN    ", "WITH    ", "XOR     ");

  special_chars_list : array(graphic_chars) of token_type :=
    ('"' => quotation,  '&' => ampersand,  ''' => tick,
    '(' => leftparen,  ')' => rightparen, ';' => semicolon,
    '+' => plus,       ',' => comma,      '-' => minus, '.' => dot,
    '/' => divide,     ':' => colon,      '*' => star, '<' => less,
    '=' => equal,      '>' => greater,    '|' => verticalbar,
    '!' => exclamation, '$' => dollar,    '%' => percent,
    '?' => question,   '@' => atsign,     '[' => leftsquare,
    ']' => rightsquare, '\' => backslash, '^' => circumflex,
    '`' => grave,      '{' => leftbrace,  '}' => rightbrace,
    '~' => tilde,      '_' => underscore, '#' => sharp,
    'A'..'Z' => other, 'a'..'z' => other, '0'..'9' => other,
    ' ' => other);

  source_line : line(1..max_length) := (others => ' ');
  token : token_type := other;
  char_position, right_paren_at : natural := 0;
  line_length : natural := 0;
  line_finished, line_needs_output : boolean := false;
  finished : boolean := false;
  char, next_ char : character := ' ';
  uc_conversion : array('a'..'z') of character :=
                        "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
  blank_identifier, current_identifier, task_analyzer_pragma_name,
    current_task_name : string(1..max_length) := (others => ' ');
  identifier_length, task_name_length : natural := 0;

  procedure get_token is

constraint_error: exception;
```

```
    word_line : string(1..max_length) := (others => ' ');
    word_position : natural := 0;
    word : word_type := (others => ' ');

    procedure get_character is
    begin

      if char_position < line_length then
        char_position := char_position + 1;
        char := next_char;
        next_char := source_line(char_position);
        if next_char in 'a'..'z' then
          next_char := uc_conversion(next_char);
        end if;
      else
        if line_finished then
          line_finished := false;
          line_needs_output := true;
        else
          line_finished := true;
        end if;
        char := next_char;
        next_char := ' ';
      end if;

    end get_character;

    procedure string_handler is
    begin
      get_character;
      get_character;
      if char = "" and next_char = "" then
        string_handler;
      end if;
    end string_handler;

  begin  -- get_token

    current_identifier := blank_identifier;

-- When a line of input has been exhausted, get a new one. When
-- the end of the file has been reached, return out.

    if line_needs_output then
      line_needs_output := false;
      --new_line (write_file);
      put_line (write_file, source_line(1..line_length));
```

```
      --new_line (write_file);
      if not end_of_file (read_file) then
        get_line (read_file, source_line, line_length);
        char_position := 1;
        next_char := source_line(char_position);
        if next_char in 'a'..'z' then
          next_char := uc_conversion(next_char);
        end if;
      else
        finished := true;
        return;
      end if;
    end if;

    token := other;

-- Suppress blanks and tabs

    while (char=' ') or (char in format_effectors)  loop
      get_character;
      if line_finished then
        exit;
      end if;
    end loop;

-- Build identifiers

    if char in letter then
      token := identifier;
      while (char in letter) or (char in digit) or (char = '_') loop
        word_position := word_position + 1;
        word_line(word_position) := char;
        get_character;
      end loop;
      current_identifier := word_line;
      identifier_length := word_position;

-- Recognize reserved words

      word := word_line(1..10);
      for index in reserved'first .. reserved'last loop
        if word = reserved_word_list(index) then
          token := index;
          exit;
        end if;
      end loop;
```

```
-- Build numbers

   elsif char in digit then
     token := numliteral;

   -- Recognize an integer or a base.

     while char in digit or char = '_' loop
       get_character;
     end loop;

   -- Recognize a real number.

     if char = '.' and next_char /= '.' then
       get_character;
       while char in digit or char = '_' loop
         get_character;
       end loop;

   -- Recognize a based integer or a based real number.

     elsif char = '#' then
       get_character;
       while char in digit or char = '_' or
                     char in letter loop
         get_character;
       end loop;
       if char ='.' and next_char /= '.' then
         get_character;
         while char in digit or char ='_' or
                       char in letter loop
           get_character;
         end loop;
       end if;
       if char = '#' then
         get_character;
       end if;
     end if;

   -- Recognize an exponent following an integer, real, or
   -- based number.

     if char = 'E' then
       get_character;
       if char = '+' or char = '-' then
         get_character;
       end if;
```

```
      while char in digit loop
        get_character;
      end loop;
    end if;

-- All other symbols

  else

    if char = ')' then
      right_paren_at := char_position - 1;
    end if;

        token := special_chars_list(char);


    get_character;

-- Recognize a character literal

    if token = tick then
      if next_char = ''' then
        get_character;
        get_character;
        token := charliteral;
      end if;
    end if;

-- Recognize a string literal

    if token = quotation then
      if char = '"' then
        get_character;
        if char = '"'  and next_char /= '"' then
          get_character;
          while char /= '"' loop
            get_character;
            if char = '"' and next_char = '"' then
              string_handler;
            end if;
          end loop;
        get_character;          -- reposition after quote
        end if;                 -- otherwise null string
      else
        while char /= '"' loop
          get_character;
          if char = '"' and next_char = '"' then
```

```
            string_handler;
          end if;
        end loop;
        get_character;          -- reposition after quote
      end if;
      token := stringliteral;
    end if;

-- Build compound delimiters

    if token in simple_delimiter then
      case token is
        when star =>
          if char = '*' then
            token := exponentiate;
          end if;
        when divide =>
          if char = '=' then
            token := notequal;
          end if;
        when minus =>
          if char = '-' then
            token := comment;
            get_character;
          end if;
        when dot =>
          if char = '.' then
            token := doubledot;
          end if;
        when colon =>
          if char = '=' then
            token := becomes;
          end if;
        when less =>
          if char = '<' then
            token := leftlabel;
          elsif char = '=' then
            token := lessequal;
          elsif char = '>' then
            token := box;
          end if;
        when equal =>
          if char = '>' then
            token := arrow;
          end if;
        when greater =>
          if char = '>' then
```

```
            token := rightlabel;
          elsif char = '=' then
            token := greaterequal;
          end if;
        when others => null;
      end case;
      if token in compound_delimiter then
        get_character;
      end if;
    end if;
  end if;
end get_token;

-- The mini tasking parser for the PAMAS front-end will generally follow
-- the basic syntax of Ada, but will usually only pay attention to detail
-- when tasking structures are being parsed.

  procedure tasking_parser is separate;

begin  -- scan
  task_analyzer_pragma_name(1..13) := "TASK_ANALYZER";
  get_line (read_file, source_line, line_length);
  char_position := 1;
  next_char := source_line(char_position);
  if next_char in 'a'..'z' then
    next_char := uc_conversion(next_char);
  end if;

  get_token;
  tasking_parser;
end scan;
```

```
--
-- ****************************************
-- Portable Ada Multitasking Analyzer System
--
--          Version 1.0
--
--  Designed, developed, and written by:
--          Robert J. Knapper
--          David O. LeVan
--             of the
-- Computer Software and Engineering Division
--     Institute for Defense Analyses
--          Alexandria, VA
--
--             11/8/88
--
-- ****************************************
--
-- The parsing phase of PAMAS is limited in its scope. The basic
-- compilation structure of an Ada program will be followed, but
-- only those features related to tasking (and this includes some
-- declarations, i.e. task types and objects thereof, entries, etc.)
-- will be considered. What this will be in effect is a "squashed"
-- grammar parsing for Ada.

-- The procedures of the "parser" will need access to each other
-- so they are declared separately as specs and bodies.

separate(pamas_front_end.scan)
procedure tasking_parser is

  type item_kind is (package_object, task_object, task_type, access_object,
                     object, renamed_object, record_type, record_object,
                     array_type, array_object);

  type symbol_table_item(length : natural);
  type task_table_item(length : natural);
  type entry_list_item(length : natural);

  type symbol_table_item_ptr is access symbol_table_item;
  type task_table_item_ptr is access task_table_item;
  type entry_list_item_ptr is access entry_list_item;

  type symbol_table_item(length : natural) is
    record
      name : string(1..length);
      kind : item_kind;
```

```
    task_reference : task_table_item_ptr;
    next : symbol_table_item_ptr;
  end record;

type task_table_item(length : natural) is
  record
    name : string(1..length);
    entry_list : entry_list_item_ptr;
    next : task_table_item_ptr;
  end record;

type entry_list_item(length : natural) is
  record
    name : string(1..length);
    entry_index, formal_part : boolean := false;
    next : entry_list_item_ptr;
  end record;

type instruments is (task_call_before, start_rendezvous, end_of_program,
                     start_accept_before, end_accept, task_end,
                     task_init, start_select_norm, end_select, task_abort,
                     task_delay_before, else_alternative,
-- module_type
                     is_file, is_package, is_procedure, is_function,
                     is_task,
-- module_modifier
                     is_normal, is_generic, is_type);

package int_io is new integer_io (integer);
package instr_io is new enumeration_io (instruments);

new_source_line, blank_line : string(1..max_length) := (others => ' ');
linked_program_unit, main_program : boolean := false;
group_number, link_number, probe_number : integer := 0;
first_item, current_item : symbol_table_item_ptr;
current_entry : entry_list_item_ptr;
current_unit : instruments := is_file;
current_modifier : instruments := is_normal;

procedure pragma_handler;
procedure compilation;
procedure compilation_unit;
procedure context_clause;
procedure subprogram_handler;
procedure subprogram_specification;
procedure package_handler;
procedure package_specification;
```

```
procedure separate_handler;
procedure subprogram_body;
procedure package_body;
procedure proper_body;
procedure with_clause;
procedure use_clause;
procedure declarative_part;
procedure sequence_of_declarations;
procedure declaration;
procedure identifier_declaration;
procedure type_declaration;
procedure subprogram_declaration;
procedure package_declaration;
procedure task_declaration;
procedure task_handler;
procedure task_body;
procedure generic_declaration;
procedure representation_clause;
procedure sequence_of_statements (group : in integer;
                                  select_alt: in BOOLEAN:=FALSE);
procedure statement (group : in integer;
                     select_alt: in BOOLEAN:=FALSE);
procedure block_statement (group : in integer);
procedure if_statement (group : in integer);
procedure case_statement (group : in integer);
procedure loop_statement (group : in integer);
procedure select_statement (group : in integer);
procedure select_alternative (group : in integer);
procedure accept_statement (group : in integer;
                inside_select : in boolean);
procedure delay_statement (group : in integer;
                           select_alt:in boolean);
procedure terminate_alternative (group : in integer);
procedure abort_statement (group : in integer);
procedure identifier_statement (group : in integer;
                                select_alt: in BOOLEAN:=FALSE);
procedure exception_handler (group : in integer);
procedure formal_part_handler;
procedure entry_index_or_formal_part (formal_part : out boolean);

procedure enter_symbol (kind : in item_kind);
procedure check_for_symbol (found : out boolean;
                found_task : out task_table_item_ptr;
                found_symbol : out symbol_table_item_ptr);
procedure write_a_probe (action : in instruments;
              task_name, entry_name : in string;
              group : in integer;
```

```
                        parameters_added: in BOOLEAN := FALSE);
procedure write_a_link_init (module_type,
                module_modifier : in instruments;
                group : in integer);
procedure add_parameters_to_entry;
procedure add_parameters_to_call (task_name,
                    entry_name : in string;
                    task_name_length,
                    entry_name_length : in natural;
                            group: in integer);

procedure pragma_handler is
begin
 while token = pragmatoken loop
   while token /= semicolon loop
     get_token;
     if current_identifier = task_analyzer_pragma_name then
       main_program := true;
     end if;
   end loop;
   get_token;
 end loop;
end pragma_handler;

procedure compilation is
begin
  loop
    put_line(write_file, "with mtd_fundamental_types; " &
                "use mtd_fundamental_types;");
    put_line(write_file, "with useful_types; use useful_types;");
    put_line(write_file, "with mtd_complex_types; " &
                "use mtd_complex_types;");
    put_line(write_file, "with mtd_tool; use mtd_tool;");
    put_line(write_file, "with link_init;");
    put_line(write_file, "with new_a_strings; use new_a_strings;");
    new_line(write_file);
    pragma_handler;
    compilation_unit;
    exit when finished;
  end loop;
end compilation;

procedure compilation_unit is
begin
  context_clause;
  case token is
    when proceduretoken | functiontoken => subprogram_handler;
```

```
    when generictoken => generic_declaration;
    when packagetoken => package_handler;
    when separatetoken => separate_handler;
    when others => null;
  end case;
end compilation_unit;

procedure context_clause is
begin
  while token = withtoken loop     -- zero or more "withs"
    with_clause;
    pragma_handler;
    while token = usetoken loop    -- zero or more "uses"
      use_clause;
      pragma_handler;
    end loop;
  end loop;
end context_clause;

procedure subprogram_handler is
begin
  subprogram_specification;
  if token = renamestoken then
    while token /= semicolon loop
      get_token;
    end loop;
    get_token;                     -- next declaration start
  end if;
  if token = istoken then
    get_token;
    if token = newtoken then
      while token /= semicolon loop
        get_token;                 -- skip to ";"
      end loop;
      get_token;                   -- next declaration start
    elsif token = separatetoken then
      get_token;
      get_token;                   -- next declaration start
    else
      subprogram_body;
    end if;
  end if;
end subprogram_handler;

procedure subprogram_specification is
begin
  if token = proceduretoken then
```

```
    current_unit := is_procedure;
  else
    current_unit := is_function;
  end if;
  get_token;                -- identifier/designator
  get_token;                -- "(", ";", is, renames, return
  if token = leftparen then
    get_token;
    formal_part_handler;        -- formal_part
    get_token;                -- ";", is, return
  end if;
  if token = returntoken then
    get_token;                -- type_mark
    get_token;                -- ".", ";", renames, or is
    while token = dot loop
      get_token;
      get_token;
    end loop;
  end if;
  if token = semicolon then
    get_token;                -- next declaration start
  end if;
end subprogram_specification;

procedure package_handler is
begin
  get_token;
  if token = bodytoken then
    get_token;
    package_body;
  else
    package_specification;
  end if;
end package_handler;

procedure package_specification is
begin
  enter_symbol (package_object);
  get_token;                -- is or renames
  if token = renamestoken then
    while token /= semicolon loop
      get_token;
    end loop;
  elsif token = istoken then
    get_token;                -- declaration start
    if token = newtoken then
      while token /= semicolon loop
```

```
        get_token;              -- skip to ";"
      end loop;
    else
      declarative_part;
      if token = privatetoken then
        get_token;
        declarative_part;
      end if;
      get_token;                -- identifier or ";"
      if token /= semicolon then
        get_token;
      end if;
    end if;
  end if;
  get_token;                    -- next declaration start
end package_specification;

procedure separate_handler is
begin
  get_token;                -- leftparen
  loop
    get_token;
    exit when token = rightparen;
  end loop;
  get_token;                -- proper body start
  proper_body;
end separate_handler;

procedure subprogram_body is
  save_unit_state : boolean := linked_program_unit;
  save_group : integer;
  save_main_program : boolean := main_program;
  save_current_unit : instruments := current_unit;
begin
  main_program := false;
  linked_program_unit := false;
  group_number := group_number + 1;
  put_line (write_file, "-#");
  int_io.put (write_file, group_number, 0);
  new_line (write_file);
  save_group := group_number;
  declarative_part;
  get_token;                    -- after begin
  sequence_of_statements (save_group);
  main_program := save_main_program;
  current_unit := save_current_unit;
  if main_program then
```

```
    if not linked_program_unit then
      linked_program_unit := true;
      write_a_link_init (current_unit, current_modifier, save_group);
    end if;
    write_a_probe (end_of_program, "", "", save_group);
  end if;
  if token = exceptiontoken then
    get_token;
    exception_handler (save_group);
  end if;
  get_token;                    -- ";" or designator
  if token /= semicolon then
    get_token;
  end if;
  linked_program_unit := save_unit_state;
  get_token;                    -- next unit start
end subprogram_body;

procedure package_body is
  save_unit_state : boolean := linked_program_unit;
  save_group : integer;
begin
  current_unit := is_package;
  get_token;                    -- is
  get_token;                    -- declaration, begin or separate
  if token = separatetoken then
    get_token;
  else
    linked_program_unit := false;
    group_number := group_number + 1;
    put_line (write_file, "--#");
    int_io.put (write_file, group_number, 0);
    new_line (write_file);
    save_group := group_number;
    declarative_part;
    if token = begintoken then
      get_token;                -- after begin
      sequence_of_statements (save_group);
      linked_program_unit := save_unit_state;
      if token = exceptiontoken then
        get_token;
        exception_handler (save_group);
      end if;
    end if;
    get_token;                  -- ";" or designator
    if token /= semicolon then
      get_token;
```

```
    end if;
  end if;
  get_token;                    -- next unit start
end package_body;

procedure proper_body is
begin
  pragma_handler;
  case token is
    when proceduretoken | functiontoken => subprogram_handler;
    when packagetoken => package_handler;
    when tasktoken => task_handler;
    when others => put_line(write_file, "-- Error in proper body");
  end case;
end proper_body;

procedure with_clause is
begin
  while token /= semicolon loop
    get_token;
  end loop;
  get_token;            -- next stmt start
end with_clause;

procedure use_clause is
begin
  while token /= semicolon loop
    get_token;
  end loop;
  get_token;            -- next stmt start
end use_clause;

procedure declarative_part is
begin
  sequence_of_declarations;
end declarative_part;

procedure sequence_of_declarations is
begin
  pragma_handler;
  if token /= begintoken and token /= endtoken
                and token /= privatetoken then
    declaration;
    sequence_of_declarations;
  end if;
end sequence_of_declarations;
```

```
procedure declaration is
begin
  case token is
    when pragmatoken      =>
      while token /= semicolon loop
        get_token;
      end loop;
      get_token;
    when identifier       => identifier_declaration;
    when typetoken |
      subtypetoken        => type_declaration;
    when proceduretoken |
      functiontoken       => subprogram_declaration;
    when packagetoken     => package_declaration;
    when tasktoken        => task_declaration;
    when generictoken     => generic_declaration;
    when usetoken         => use_clause;
    when fortoken         => representation_clause;
    when others           =>
      put_line(write_file, "-- No more declarations");
  end case;
end declaration;

procedure identifier_declaration is
  old_current_item : symbol_table_item_ptr := current_item;
  new_current_item : symbol_table_item_ptr;
  symbol_found : boolean := false;
  item_found : task_table_item_ptr;
  table_item : symbol_table_item_ptr;
begin
  enter_symbol (task_object);
  new_current_item := current_item;
  get_token;              -- "," or ":"
  while token = comma loop
    get_token;            -- identifier
    enter_symbol (task_object);
    get_token;            -- "," or ":"
  end loop;
  get_token;              -- constant, exception, array,
                          -- subtype_indication
  if token = exceptiontoken then
    get_token;            -- ";"
  else
    if token = constanttoken then
      get_token;          -- ":=", array, or subtype_indication
      if token = becomes then
        while token /= semicolon loop
```

```
      get_token;        -- expression
    end loop;
  end if;
end if;
if token = identifier then
  check_for_symbol (symbol_found, item_found, table_item);
  get_token;          -- ".", ";", ":=" or "("
  if token = leftparen then
    get_token;
    formal_part_handler;
    get_token;
  end if;
  if token = renamestoken then
    while token /= semicolon loop
      get_token;
    end loop;
  end if;
  if token = rangetoken then
    while token /= semicolon loop
      get_token;
    end loop;
  end if;
  if token = becomes then
    while token /= semicolon loop
      get_token;        -- universal_static_expression
    end loop;
  end if;
elsif token = arraytoken then
  get_token;          -- "("
  get_token;
  formal_part_handler;
  get_token;          -- of
  get_token;
  check_for_symbol (symbol_found, item_found, table_item);
  get_token;          -- ".", ";", ":=" or "("
  while token /= semicolon loop
    get_token;
  end loop;
end if;
if symbol_found then
  while new_current_item /= null loop
    new_current_item.task_reference := item_found;
    new_current_item := new_current_item.next;
  end loop;
else
  if old_current_item /= null then
    current_item := old_current_item;
```

```
        current_item.next := null;
      else
        first_item := null;
        current_item := null;
      end if;
    end if;
  end if;
  get_token;              -- next declaration start
end identifier_declaration;

procedure type_declaration is
  old_current_item : symbol_table_item_ptr := current_item;
  symbol_found : boolean;
  item_found : task_table_item_ptr;
  table_item : symbol_table_item_ptr;
begin
  get_token;              -- identifier
  enter_symbol (task_type);
  get_token;              -- is, "(", or ";"
  if token = leftparen then     -- discrimnant part
    get_token;
    formal_part_handler;
    get_token;            -- is or ";"
  end if;
  if token /= semicolon then
    get_token;            -- type_definition
    case token is
      when newtoken |           -- derived type
         leftparen |          -- enumeration type
         rangetoken |
         numliteral |         -- inte   type
         digitstoken |        -- float type
         deltatoken  |        -- fixed type
         limitedtoken |
         privatetoken =>      -- private type
        while token /= semicolon loop
          get_token;
        end loop;
      when arraytoken =>
        get_token;          -- "("
        get_token;
        formal_part_handler;
        get_token;          -- of
        get_token;          -- component_subtype_indication
        check_for_symbol (symbol_found, item_found, table_item);
        get_token;          -- ".", ";", ":=" or "("
        while token /= semicolon loop
```

```
          get_token;
        end loop;
      when accesstoken =>
        get_token;          -- identifier
        check_for_symbol (symbol_found, item_found, table_item);
        get_token;          -- ".", ";" or ":="
        while token /= semicolon loop
          get_token;        -- skip to ";"
        end loop;
      when recordtoken =>
        get_token;          -- first component
        while token /= recordtoken loop
          get_token;        -- all components
        end loop;
        get_token;          -- ";"
      when identifier =>        -- subtype defintion only
        get_token;
        while token /= semicolon loop
          get_token;        -- constraint
        end loop;
      when others => put_line(write_file, "Error in type");
    end case;
    if symbol_found then
      current_item.task_reference := item_found;
    else
      if old_current_item /= null then
        current_item := old_current_item;
        current_item.next := null;
      else
        first_item := null;
        current_item := null;
      end if;
    end if;
  end if;
  get_token;              -- next declaration start
end type_declaration;

procedure subprogram_declaration is
begin
  subprogram_handler;
end subprogram_declaration;

procedure package_declaration is
begin
  package_handler;
end package_declaration;
```

```
procedure task_declaration is
begin
  task_handler;
end task_declaration;

procedure task_handler is
  current_task : task_table_item_ptr;
  current_entry : entry_list_item_ptr;
  task_type_declaration, formal_part : boolean := false;
begin
  get_token;              – identifier, type, or body
  if token = bodytoken then
    get_token;            – identifier
    task_body;
  else
    if token = typetoken then
      get_token;          – identifier;
      enter_symbol (task_type);
    else
      enter_symbol (task_object);
    end if;
    get_token;            – ";" or is
    if token /= semicolon then
      current_task := new task_table_item(current_item.length);
      current_task.name := current_item.name;
      current_item.task_reference := current_task;
      get_token;          – first entry or rep clause
      pragma_handler;
      while token = entrytoken loop
        get_token;        – entry name;
        if current_task.entry_list = null then
          current_task.entry_list :=
                  new entry_list_item(identifier_length);
          current_entry := current_task.entry_list;
        else
          current_entry.next :=
                  new entry_list_item(identifier_length);
          current_entry := current_entry.next;
        end if;
        current_entry.name := current_identifier(1..identifier_length);
        get_token;        – ";" or "("
        while token = leftparen loop
          get_token;
          entry_index_or_formal_part (formal_part);
          if formal_part then
            current_entry.formal_part := true;
            while token /= rightparen loop
```

```
            get_token;
          end loop;
          add_parameters_to_entry;
        else
          current_entry.entry_index := true;
        end if;
        get_token;
      end loop;
      get_token;          -- entry, for, pragma, or end
      pragma_handler;
    end loop;
    while token /= endtoken loop
      get_token;          -- skip to end
    end loop;
    get_token;            -- ";" or identifier
    if token /= semicolon then
      get_token;          -- ";"
    end if;
  end if;
  get_token;              -- next declaration start
 end if;
end task_handler;

procedure task_body is
 save_unit_state : boolean := linked_program_unit;
 save_group : integer;
 symbol_found : boolean;
 item_found : task_table_item_ptr;
 table_item : symbol_table_item_ptr;
 task_body_for_type : boolean := false;
begin
 current_task_name := current_identifier;
 task_name_length := identifier_length;
 check_for_symbol (symbol_found, item_found, table_item);
 if table_item.kind = task_type then
   task_body_for_type := true;
 end if;
 get_token;               -- is
 get_token;               -- declaration, begin, or separate
 if token = separatetoken then
   get_token;
 else
   linked_program_unit := true;
   group_number := group_number + 1;
   put_line (write_file, "-#");
   int_io.put (write_file, group_number, 0);
   new_line (write_file);
```

```
    if task_body_for_type then
      write_a_link_init (is_task, is_type, group_number);
    else
      write_a_link_init (is_task, is_normal, group_number);
    end if;
    save_group := group_number;
    declarative_part;
    get_token;                  -- after begin
    sequence_of_statements (save_group);
    write_a_probe (task_end, "", "", save_group);
    linked_program_unit := save_unit_state;
    if token = exceptiontoken then
      get_token;
      exception_handler (save_group);
    end if;
    get_token;                  -- ";" or designator
    if token /= semicolon then
      get_token;                -- ";"
    end if;
  end if;
  get_token;                    -- next unit start
end task_body;

procedure generic_declaration is
begin
  get_token;
  pragma_handler;
  while (token /= proceduretoken) and
      (token /= functiontoken) and
      (token /= packagetoken) loop
    if token = withtoken then
      while token /= semicolon loop
        get_token;
      end loop;
    end if;
    get_token;
  end loop;
  case token is
    when proceduretoken | functiontoken => subprogram_handler;
    when packagetoken => package_handler;
    when others => put_line(write_file, "-- Error in generic dec");
  end case;
end generic_declaration;

procedure representation_clause is
begin
  while token /= semicolon loop
```

```
    get_token;              -- rep clause
    if token = recordtoken then
      get_token;
      while token /= recordtoken loop
        get_token;
      end loop;
    end if;
  end loop;
  get_token;                -- next declaration
end representation_clause;

procedure sequence_of_statements (group : in integer;
                                  select_alt: in BOOLEAN:= FALSE) is
begin
  pragma_handler;
  statement (group, select_alt);
  if token in statement_tokens or token = leftlabel
                or token = pragmatoken then
    sequence_of_statements (group);
  end if;
end sequence_of_statements;

procedure statement (group : in integer;
                     select_alt: in BOOLEAN:=FALSE) is
begin
  while token = leftlabel loop
    get_token;              -- label name
    get_token;              -- right label
    get_token;              -- left label or statement
  end loop;
  case token is
    when nulltoken | exittoken | returntoken |
       gototoken | raisetoken => while token /= semicolon loop
                       get_token;
                     end loop;
                     get_token; -- next statement
    when iftoken => if_statement (group);
    when casetoken => case_statement (group);
    when looptoken | whiletoken | fortoken => loop_statement (group);
    when declaretoken | begintoken => block_statement (group);

-- task statements

    when selecttoken => select_statement (group);
    when accepttoken => accept_statement (group, false);
    when delaytoken  => delay_statement (group, false);
    when aborttoken  => abort_statement (group);
```

```
    when identifier  => identifier_statement (group, select_alt);

                when ortoken | elsetoken => null;  -- if or select stmt
    when others => put_line (write_file, "-- No more statements");
  end case;
end statement;

procedure block_statement (group : in integer) is
begin
  if token = declaretoken then
    get_token;                  -- declaration start
    declarative_part;
  end if;
  get_token;                    -- statement start
  sequence_of_statements (group);
  if token = exceptiontoken then
    get_token;
    exception_handler (group);
  end if;
  get_token;                    -- ";" or identifier
  if token /= semicolon then
    get_token;
  end if;
  get_token;                    -- next statement start
end block_statement;

procedure if_statement (group : in integer) is
  procedure and_then_or_else_handler is
  begin
    get_token;
    if token = thentoken or token = elsetoken then
      get_token;
    end if;
  end and_then_or_else_handler;
begin
  while token /= thentoken loop
    get_token;                  -- skip to then
    if token = andtoken or token = ortoken then
      and_then_or_else_handler;
    end if;
  end loop;
  get_token;                    -- statement start
  sequence_of_statements (group);
  pragma_handler;
  while token = elsiftoken loop
    while token /= thentoken loop
      get_token;                -- skip to then
```

```
      if token = andtoken or token = ortoken then
        and_then_or_else_handler;
      end if;
    end loop;
    get_token;                  -- statement start
    sequence_of_statements (group);
  end loop;
  pragma_handler;
  if token = elsetoken then
    get_token;                  -- statement start
    sequence_of_statements (group);
  end if;
  get_token;                    -- if
  get_token;                    -- ";"
  get_token;                    -- next statement start
end if_statement;

procedure case_statement (group : in integer) is
begin
  while token /= istoken loop
    get_token;                  -- skip to is
  end loop;
  get_token;                    -- first when
  pragma_handler;
  while token = whentoken loop
    while token /= arrow loop
      get_token;                -- skip to "=>"
    end loop;
    get_token;                  -- statement start
    sequence_of_statements (group);
  end loop;
  get_token;                    -- case
  get_token;                    -- ";"
  get_token;                    -- next statement start
end case_statement;

procedure loop_statement (group : in integer) is
begin
  while token /= looptoken loop
    get_token;                  -- skip to loop
  end loop;
  get_token;                    -- statement start
  sequence_of_statements (group);
  while token /= semicolon loop
    get_token;
  end loop;
  get_token;                    -- next statement start
```

```
end loop_statement;

procedure select_statement (group : in integer) is
  save_group : integer := group;
begin
  write_a_probe (start_select_norm, "", "", group);
  get_token;                    -- first select alternative
  select_alternative (group);
  pragma_handler;
  while token = ortoken loop
    get_token;                  -- next select alternative
    select_alternative (group);
    pragma_handler;
  end loop;
  if token = elsetoken then
    get_token;                  -- statement start
    sequence_of_statements (group);
  end if;
  get_token;                    -- select
  get_token;                    -- ";"
  get_token;                    -- next statement start
  write_a_probe (end_select, "", "", save_group);
end select_statement;

procedure select_alternative (group : in integer) is
begin
  if token = whentoken then
    while token /= arrow loop
      get_token;                -- skip to "=>"
    end loop;
    get_token;                  -- accept, delay, or terminate
  end if;
  case token is
    when accepttoken   => accept_statement (group, true);
                          sequence_of_statements (group);
    when delaytoken    => delay_statement (group, true);
                          sequence_of_statements (group);
    when terminatetoken => terminate_alternative (group);
    when identifier    => sequence_of_statements (group, TRUE);
    when others        => put_line(write_file, "-- Error in select");
  end case;
end select_alternative;

procedure accept_statement (group : in integer;
                            inside_select : in boolean) is
  formal_part, parameters_added : boolean := false;
  task_name : string(1..task_name_length) :=
```

```
            current_task_name(1..task_name_length);
  current_entry_name : string(1..max_length) := (others => ' ');
  entry_name_length : natural := 0;
  save_group : integer := group;
begin
  get_token;                    -- entry name
  current_entry_name (1..identifier_length) :=
              current_identifier(1..identifier_length);
  entry_name_length := identifier_length;
  if not inside_select then
    write_a_probe (start_accept_before, "", "", group);
  end if;
  get_token;                    -- "(", ";", do
  while token = leftparen loop      -- index or formal part
    get_token;
    entry_index_or_formal_part (formal_part);
    if formal_part then
      while token /= rightparen loop
        get_token;
      end loop;
      add_parameters_to_entry;
          parameters_added := TRUE;
    end if;
    get_token;                  -- "(", ";", do
  end loop;
  if token = dotoken then
    get_token;                  -- statement start
    write_a_probe (start_rendezvous,
              current_task_name(1..task_name_length),
              current_entry_name(1..entry_name_length), group,
                        parameters_added);
    sequence_of_statements (group);
    write_a_probe (end_accept, "", "", save_group, parameters_added);
    get_token;                  -- entry_name or ";"
    if token /= semicolon then
      get_token;                -- ";"
    end if;
  end if;
  get_token;                    -- next statement start
end accept_statement;

procedure delay_statement (group : in integer; select_alt:in boolean) is
begin
  if not select_alt then
       write_a_probe (task_delay_before, "", "", group);
  end if;
  while token /= semicolon loop
```

```
    get_token;              -- skip to ";"
  end loop;
  get_token;                -- next statement start
end delay_statement;

procedure terminate_alternative (group : in integer) is
begin
  get_token;                -- ";"
  get_token;                -- next statement start
end terminate_alternative;

procedure abort_statement (group : in integer) is
begin
  write_a_probe (task_abort, "", "", group);
  while token /= semicolon loop
    get_token;              -- skip to ";"
  end loop;
  get_token;                -- next statement start;
end abort_statement;

procedure identifier_statement (group : in integer;
                                select_alt: in BOOLEAN:=FALSE) is
  symbol_found : boolean;
  item_found : task_table_item_ptr;
  table_item : symbol_table_item_ptr;
  task_name, entry_name : string(1..max_length);
  task_name_length, entry_name_length : natural;
  formal_part : boolean := false;
  entry_found : entry_list_item_ptr;

  procedure find_entry is
    entry_item : entry_list_item_ptr := item_found.entry_list;
  begin
    while entry_item /= null loop
      if entry_item.length = entry_name_length then
        if entry_item.name = entry_name(1..entry_name_length) then
          entry_found := entry_item;
          return;
        end if;
      end if;
      entry_item := entry_item.next;
    end loop;
  end find_entry;

begin
  check_for_symbol (symbol_found, item_found, table_item);
  if symbol_found then
```

```
    task_name := current_identifier;
    task_name_length := identifier_length;
    if not linked_program_unit then
      linked_program_unit := true;
      write_a_link_init (current_unit, current_modifier, group);
    end if;
    while token /= dot loop
      get_token;                    -- "."
      if token = becomes then       -- handle assignment
        while token /= semicolon loop
          get_token;
        end loop;
        get_token;                  -- next statement start
        return;
      end if;
    end loop;
    get_token;                      -- entry name
       if not select_alt then
      write_a_probe (task_call_before, table_item.name,
            current_identifier (1..identifier_length), group);
       end if;
    entry_name := current_identifier;
    entry_name_length := identifier_length;
    find_entry;
    get_token;                      -- "(", ";"
    if token = leftparen then       -- index or formal part
      get_token;
      entry_index_or_formal_part (formal_part);
      if entry_found.formal_part and not entry_found.entry_index then
        add_parameters_to_call (task_name(1..task_name_length),
                        entry_name(1..entry_name_length),
                        task_name_length, entry_name_length, group);
      elsif entry_found.formal_part and entry_found.entry_index then
        get_token;
        get_token;
        entry_index_or_formal_part (formal_part);
        add_parameters_to_call (task_name(1..task_name_length),
                        entry_name(1..entry_name_length),
                        task_name_length, entry_name_length, group);
      end if;
    end if;
  else
    get_token;
    if token = colon then
      get_token;
      case token is
        when looptoken | whiletoken | fortoken =>
```

```
                    loop_statement (group);
        when begintoken | declaretoken => block_statement (group);
        when others => put_line(write_file, "Error in block");
      end case;
      return;
    end if;
  end if;
  while token /= semicolon loop
    get_token;              -- skip to ";"
  end loop;
  get_token;                -- next statement start
end identifier_statement;

procedure exception_handler (group : in integer) is
begin
  while token = whentoken loop
    while token /= arrow loop
      get_token;            -- skip to "=>"
    end loop;
    get_token;              -- statement start
    sequence_of_statements (group);
  end loop;
end exception_handler;

procedure formal_part_handler is
begin
  loop
    if token = leftparen then
      get_token;
      formal_part_handler;
    end if;
    get_token;
    exit when token = rightparen;
  end loop;
end formal_part_handler;

procedure entry_index_or_formal_part (formal_part : out boolean) is
begin
  loop
    if token = leftparen then
      get_token;
      entry_index_or_formal_part (formal_part);
    end if;
    get_token;
    if token = colon then
      formal_part := true;
      return;
```

```
    end if;
    exit when token = rightparen;
  end loop;
  formal_part := false;
end entry_index_or_formal_part;

procedure enter_symbol (kind : in item_kind) is
begin
  if first_item = null then
    first_item := new symbol_table_item(identifier_length);
    current_item := first_item;
  else
    current_item.next := new symbol_table_item(identifier_length);
    current_item := current_item.next;
  end if;
  current_item.kind := kind;
  current_item.name := current_identifier(1..identifier_length);
  current_item.next := null;
end enter_symbol;

procedure check_for_symbol (found : out boolean;
                            found_task : out task_table_item_ptr;
                            found_symbol : out symbol_table_item_ptr) is
  item_ptr : symbol_table_item_ptr := first_item;
  identifier_name : string(1..max_length) := current_identifier;
begin
  found := false;
  found_task := null;
  while item_ptr /= null loop
    if item_ptr.length = identifier_length and then
       item_ptr.name = identifier_name(1..identifier_length) then
      if item_ptr.kind = package_object then
        get_token;
        get_token;
        identifier_name := current_identifier;
      elsif item_ptr.kind = task_object  or
                item_ptr.kind = task_type then
          found := true;
          found_task := item_ptr.task_reference;
          found_symbol := item_ptr;
          return;
      end if;
    end if;
    item_ptr := item_ptr.next;
  end loop;
end check_for_symbol;
```

```
procedure write_a_probe (action : in instruments;
                task_name, entry_name : in string;
                group : in integer;
                        parameters_added: in BOOLEAN := FALSE) is
begin
  probe_number := probe_number + 1;
  new_line (instr_file);
  put (instr_file, "   probe_");
  int_io.put (instr_file, probe_number, 0);
  put_line (instr_file, "_first_time : boolean := true;");
  put (instr_file, "   probe_");
  int_io.put (instr_file, probe_number, 0);
  put_line (instr_file, "_id : p_id := -1;");
  put (instr_file, "   procedure probe_");
  int_io.put (instr_file, probe_number, 0);
  put_line (instr_file, " is new probe(");
  put (instr_file, "      p_name => ""probe_");
  int_io.put (instr_file, probe_number, 0);
  put_line (instr_file, """",");
  put (instr_file, "      t_action => ");
  instr_io.put (instr_file, action);
  put_line (instr_file, ",");
  put_line (instr_file, "      p_path => "".""""");
  put_line (instr_file, "      );");
  new_line (write_file);
  put (write_file, "      probe_");
  int_io.put (write_file, probe_number, 0);
  put_line (write_file, "(");
  put (write_file, "         link_task => " & "init_link_");
  int_io.put (write_file, group, 0);
  put_line (write_file, ".module_dynamic_info.link_task,");
  put (write_file, "         module => " & "init_link_");
  int_io.put (write_file, group, 0);
  put_line (write_file, ".module_dynamic_info.id,");
  put (write_file, "         id => probe_");
  int_io.put (write_file, probe_number, 0);
  put_line (write_file, "_id,");
  put (write_file, "         first_time => probe_");
  int_io.put (write_file, probe_number, 0);
  put (write_file, "_first_time");

  if (action = start_rendezvous or action = end_accept) and parameters_added then
    put_line (write_file, ",");

    put (write_file, "         task_call_name => ");

    put_line (write_file, "to_a(task_called_as),");
```

```
    put (write_file, "      parent_module => " &
                      "module_identity");

  end if;
  if action = task_call_before then
    put_line (write_file, ",");

    put (write_file, "      task_call_name => ");

    put (write_file, "to_a(""" & task_name & "." & entry_name & """""");
    put (write_file, ")");

  end if;
  if action = start_rendezvous then

    put_line (write_file, ",");
    put (write_file, "      number_queued => " & task_name &
              "." & entry_name & "'COUNT");
  end if;

    new_line (write_file);
    put_line (write_file, "      );");
    return;

end write_a_probe;

procedure write_a_link_init (module_type,
                  module_modifier : in instruments;
                  group : in integer) is
begin
  put_line (instr_file, "--#");
  int_io.put (instr_file, group, 0);
  new_line (instr_file, 2);
  put (instr_file, "package instruments_");
  int_io.put (instr_file, group, 0);
  put_line (instr_file, " is");
  put (instr_file, "  package init_link_");
  int_io.put (instr_file, group, 0);
  put_line (instr_file, " is new link_init(");
  put (instr_file, "    m_group => ");
  int_io.put (instr_file, group, 0);
  put_line (instr_file, ",");
  put_line (instr_file, "    f_name => """ &
       main_file_name(1..main_name_length) & """,");
  put_line (instr_file, "    m_name => ""."",");
  put (instr_file, "    m_type => ");
  instr_io.put (instr_file, module_type);
```

```
  put_line (instr_file, ",");
  put (instr_file, "     m_modifier => ");
  instr_io.put (instr_file, module_modifier);
  new_line (instr_file);
  put_line (instr_file, "     );");
  put (instr_file, "  use init_link_");
  int_io.put (instr_file, group, 0);
  put_line (instr_file, ";");
end write_a_link_init;

procedure int_to_string(
        number:in integer;
        string_rep:in out string;
        start: out natural) is

        layout_error: exception;

 begin
        string_rep(1..string_rep'LENGTH) := (others=> ' '); -- blank fill
        int_io.put(string_rep,number); -- convert number to string
        start:= string_rep'LENGTH;

        for first_char in 1..string_rep'LENGTH loop
          if string_rep(first_char) /= ' ' then
                start:= first_char;
                exit;
          end if;
        end loop;

 exception
        when others=>
          start:=0;
 end int_to_string;

procedure add_parameters_to_entry is
begin
  new_source_line := blank_line;
  new_source_line(1..right_paren_at-1) :=
                        source_line(1..right_paren_at-1);
  new_source_line(right_paren_at) := ';';
  new_source_line(right_paren_at+1) := ASCII.CR;
  new_source_line(right_paren_at+2) := ASCII.LF;
  new_source_line(right_paren_at+3..right_paren_at+30) :=
                        "module_identity : module_id;";
  new_source_line(right_paren_at+32..right_paren_at+55):=
                          "task_called_as : string;";
  new_source_line(right_paren_at+55..right_paren_at +
```

```
            55+line_length-right_paren_at) :=
             source_line(right_paren_at..line_length);
  char_position := char_position + 55;
  line_length := line_length + 55;
  source_line := new_source_line;
end add_parameters_to_entry;

procedure add_parameters_to_call (task_name,
                    entry_name : in string;
                    task_name_length,
                    entry_name_length : in natural;
                                group: in integer) is

  new_position, first_nonblank : natural;
  group_string: string(1..4) := (others=> ' ');  -- holds thousands of groups

begin
  new_source_line := blank_line;
  new_source_line(1..right_paren_at-1) :=
                    source_line(1..right_paren_at-1);
  new_source_line(right_paren_at) := ',';
  new_source_line(right_paren_at+1) := ASCII.CR;
  new_source_line(right_paren_at+2) := ASCII.LF;

  int_to_string(group,group_string, first_nonblank);

  new_source_line(right_paren_at+3..right_paren_at+12) :=
                "init_link ";
  new_position := right_paren_at+13+group_string'LENGTH-first_nonblank;
  new_source_line(right_paren_at+13..new_position) :=
      group_string(first_nonblank..group_string'LENGTH);
  new_source_line(new_position+1..new_position+1+23):=
      ".module_dynamic_info.id,";
  new_source_line(new_position+25) := "";
  new_source_line(new_position+26..new_position+26+
          task_name_length-1) := task_name;
  new_position := new_position+26+task_name_length;
  new_source_line(new_position) := '.';
  new_source_line(new_position+1..new_position+1+
          entry_name_length-1) := entry_name;
  new_position := new_position+1+entry_name_length;
  new_source_line(new_position) := "";
  new_source_line(new_position+1..new_position+1+
           line_length-right_paren_at) :=
            source_line(right_paren_at..line_length);
  char_position := char_position + new_position + 1;
  line_length := line_length + new_position + 1;
```

```
    source_line := new_source_line;
  end add_parameters_to_call;

begin     -- tasking_parser

  compilation;

end tasking_parser;
```

```
--
-- ****************************************
-- Portable Ada Multitasking Analyzer System
--
--          Version 1.0
--
--  Designed, developed, and written by:
--          Robert J. Knapper
--          David O. LeVan
--             of the
-- Computer Software and Engineering Division
--     Institute for Defense Analyses
--          Alexandria, VA
--
--             11/8/88
--
-- ****************************************
--
-- The weak link in the front-end. Slows the front-end
-- down since the file is "reset" for each search and merge
-- operation.

separate(pamas_front_end)
procedure merge is

   source_line, instr_line, group, blank_line :
                    string(1..max_length) := (others => ' ');
   source_length, instr_length, group_length : natural;
   merge_it : boolean;

begin
  while not end_of_file (merge1_file) loop
    get_line (merge1_file, source_line, source_length);
    if source_line(1..3) = "--#" then
      get_line (merge1_file, group, group_length);
      merge_it := false;
      while not end_of_file (merge2_file) loop
        get_line (merge2_file, instr_line, instr_length);
        if instr_line(1..3) = "--#"  or
                          end_of_file (merge2_file) then
          if merge_it then
            if end_of_file (merge2_file) then
              put_line (merged_file, instr_line(1..instr_length));
            end if;
            put (merged_file, "end instruments_");
            put (merged_file, group(1..group_length));
            put_line (merged_file, ";");
```

```
            put (merged_file, "use instruments_");
            put (merged_file, group(1..group_length));
            put_line (merged_file, ";");
            new_line (merged_file);
            reset (merge2_file, in_file);
            exit;
          else
            instr_line := blank_line;
            if end_of_file (merge2_file) then
              reset (merge2_file);
              exit;
            end if;
            get_line (merge2_file, instr_line, instr_length);
            if group = instr_line then
              merge_it := true;
            end if;
          end if;
        else
          if merge_it then
            put_line (merged_file, instr_line(1..instr_length));
          end if;
        end if;
      end loop;
    else
      put_line (merged_file, source_line(1..source_length));
    end if;
    group := blank_line;
    source_line := blank_line;
  end loop;
end merge;
```

B.2 Analyzer Control System

```
--
-- ****************************************
-- Portable Ada Multitasking Analyzer System
--
--          Version 1.0
--
--  Designed, developed, and written by:
--          David O. LeVan
--          Robert J. Knapper
--             of the
-- Computer Software and Engineering Division
--    Institute for Defense Analyses
--          Alexandria, VA
--
--             11/8/88
--
-- ****************************************
--
package useful_types is
--
-- This package contains general type definitions for use by any other package
--
subtype TEXT is POSITIVE range 1..80 ;-- range for character string length

type STRING_REC(len: TEXT:=1) -- Variable length string capability
is record
  s:STRING(1..len);
end record;

type A_STRING is access STRING_REC;

end useful_types;
```

```
--
-- ****************************************
-- Portable Ada Multitasking Analyzer System
--
--         Version 1.0
--
--  Designed, developed, and written by:
--         David O. LeVan
--        Robert J. Knapper
--           of the
-- Computer Software and Engineering Division
--    Institute for Defense Analyses
--         Alexandria, VA
--
--           11/8/88
--
-- ****************************************
--

with useful_types; use useful_types;

package mtd_fundamental_types is
--
-- Fundamental type definitions for use in the multi-tasking debugger system
--

type STATE_CONTROL is (
--
-- These are the states of the probe controlled by the monitor
--
  nop,
  terminate_task,
  raise_exception,
  delay_task);

type TASK_ACTION is (
--
-- These are the task actions that the probes monitor.  This information
-- is passed to the monitor by each probe.
--

    task_init,       -- task initialization
    start_select_norm, -- start of select block for normal
                -- entry calls
    start_select_cond, -- start of select block for conditional
                -- entry calls
    start_select_timed, -- start of select block for timed entry
```

```
                -- calls
    end_select,       -- end of select block
    task_terminate,   -- terminate a task
    task_abort,       -- abort a task
    raise_exception,  -- raise an exception
    task_delay_before, -- delay a task
       task_delay_after,
    task_end,         -- end of task execution
    task_call_before, -- call to a task
       task_call_after,
    else_alternative, -- else part in a select statement
    start_accept_before, -- start of an accept block (may or may not have code)
       start_accept_after, -- accept is a sync point only, no rendezvous code
    start_rendezvous, -- start of the rendezvous code
    end_accept,       -- end of an accept block rendezvous code
       end_of_program);  -- normal end of entire program


type STATE_CONTROL_INFO
--
-- This is supplemental information for state control of the probe
--
is record
  sel_except: INTEGER; -- Select an exception to raise
  delay_val: DURATION; -- Value to use in delay
end record;

type LINK_STATUS is (
--
-- Used to report operative status of Link between link task and its parent
  parent_ok,
  parent_not_there);

type LINK_ACTION is (
--
-- These are the possible actions the link task can take as commanded
-- by the monitor task.
--
  terminate_link,
  pass_info,
  pass_info_terminate);


subtype G_ID is INTEGER; -- Base module group type
subtype P_ID is INTEGER;-- Base probe id
subtype L_ID is INTEGER;-- Base link id
```

```
type TYPE_OF_MODULE is (
--
-- These are the types of logical modules in the system.
-- Currently, only file, package, and task are used.
--
   is_file,
   is_package,
   is_procedure,
   is_function,
   is_task);

type TYPE_OF_MODULE_MODIFIER is (
--
-- This type is used to modify the meaning of the type_of_module
-- The information is most useful to the user.
--
   is_normal,
   is_generic,
   is_type);

end mtd_fundamental_types;  -- End of the package
```

```
--
-- ****************************************
-- Portable Ada Multitasking Analyzer System
--
--          Version 1.0
--
--  Designed, developed, and written by:
--          David O. LeVan
--          Robert J. Knapper
--             of the
-- Computer Software and Engineering Division
--     Institute for Defense Analyses
--          Alexandria, VA
--
--             11/8/88
--
-- ****************************************
--
with mtd_fundamental_types; use mtd_fundamental_types;
with useful_types; use useful_types;

package mtd_link is

task type LINK is
--
-- This forms the link between the original instrumented tasks and the monitor
--
   entry signal_from_monitor(
         proc_action: STATE_CONTROL;
         selection: STATE_CONTROL_INFO;
         l_action: LINK_ACTION;
         l_status:in out LINK_STATUS);
   --
   -- The link task waits on rendevous here until the monitor signals to it
   -- thus releasing it.
   --
   -- proc_action: action to pass on to the probe
   -- selection: extra info for the probe to execute its action
   -- l_action: action the link task is to perform
   -- l_status: status of the link with the parent task
   --

   entry signal_to_process(
         proc_action: out STATE_CONTROL;
         selection: out STATE_CONTROL_INFO);
   --
   -- The probe in the instrumented task has been waiting on rendevous here.  It is
```

```
  -- released when a call is made to the signal_from_monitor entry.
  --
  -- proc_action: action info for probe
  -- selection: extra info for probe's action
  --

end LINK;

type A_LINK is access LINK;

end mtd_link;
```

```
--
-- ****************************************
-- Portable Ada Multitasking Analyzer System
--
--          Version 1.0
--
--   Designed, developed, and written by:
--          David O. LeVan
--          Robert J. Knapper
--             of the
-- Computer Software and Engineering Division
--     Institute for Defense Analyses
--          Alexandria, VA
--
--             11/8/88
--
-- ****************************************
--
package body mtd_link is

task body link is

t_proc_action: STATE_CONTROL;
t_selection: STATE_CONTROL_INFO;
t_l_action: LINK_ACTION:=pass_info;

begin
--
-- Purpose is to wait until the monitor signals this task.  A probe may be waiting
-- on the signal to process and this causes the code containing the probe to be in
-- a paused condition.  The select is necessary to show that the task is to wait for
-- a rendezvous and do nothing else.
--
loop
  select

     accept signal_from_monitor(
          proc_action: STATE_CONTROL;
          selection: STATE_CONTROL_INFO;
          l_action: LINK_ACTION;
          l_status:in out LINK_STATUS) do

          l_status:=parent_ok; -- we assume
-- grab the monitor's parameters
          t_proc_action := proc_action;
          t_selection := selection;
          t_l_action := l_action;
```

```
            if (t_l_action=pass_info) or
                  (t_l_action= pass_info_terminate) then

                  select
                    accept signal_to_process(
                    proc_action: out STATE_CONTROL;
                    selection: out STATE_CONTROL_INFO) do
-- pass the parameters from the monitor to the probe
                          proc_action := t_proc_action;
                          selection := t_selection;

                    end signal_to_process;
                  or
                    delay 10.0;
                    l_status:= parent_not_there;

                  end select;
            end if;

         end signal_from_monitor;

   or
         delay 60.0;-- nice long wait
   end select;

   exit when (t_l_action=pass_info_terminate) or (t_l_action= terminate_link);

end loop;

end link;

end mtd_link;
```

```
--
-- ******************************************
-- Portable Ada Multitasking Analyzer System
--
--          Version 1.0
--
--  Designed, developed, and written by:
--          David O. LeVan
--          Robert J. Knapper
--             of the
-- Computer Software and Engineering Division
--     Institute for Defense Analyses
--          Alexandria, VA
--
--             11/8/88
--
-- ******************************************
--

with useful_types; use useful_types;
with mtd_fundamental_types; use mtd_fundamental_types;
with mtd_link; use mtd_link;

package mtd_complex_types is

type FLAG
--
-- Record to hold monitor activity for reporting to the user
--
is record
   active:BOOLEAN:=FALSE;
   count:INTEGER:=0;
end record;

type MODULE_ID
--
-- Record to contain the ID for a module.  The group is for the unique
-- template while the link ID id for that particular instantiation
-- of that template and the corresponding link.
--
is record
   g_num: G_ID:= -1;
   l_num: L_ID:= -1;
end record;

type LOGICAL_MODULE
--
```

```
-- Record to hold static type of information for a logical
-- module.
--
is record
   module_name: STRING_REC;
   module_type: TYPE_OF_MODULE;
   modifier: TYPE_OF_MODULE_MODIFIER;
end record;


type PROBE_INFO_S
--
-- Static information for probes.  Passed to monitor by the probe
-- login process.
--
is record
   probe_name:STRING_REC;
   action: TASK_ACTION;
   probe_path:STRING_REC;
end record;

type PROBE_INFO_D
--
-- Dynamic information for probes.  Tracks changing probe information.
--
is record
   action: STATE_CONTROL:= nop;
   control_info: STATE_CONTROL_INFO;
   waiting: BOOLEAN:= FALSE;
   parent: MODULE_ID;
   task_call_name: STRING_REC;
end record;

type MODULE_INFO_S
--
-- Static information for tasks
--
is record
   file_name: STRING_REC;
   module:LOGICAL_MODULE;
end record;


type CURRENT_CHILD_INFO is
--
-- Used to indicate when a task entry has been called by a particular
-- parent process (execution thread parent).  The probe ID is
```

```
  – usually the probe placed after the corresponding accept statement.
  –
  record
    is_executing:BOOLEAN:=FALSE;
    at_probe:P_ID:=-1;
  end record;

  type MODULE_INFO_D
  –
  – Dynamic information for the logical modules in the system
  –
  is record
    link_task: A_LINK;
    id: MODULE_ID;
    action: LINK_ACTION;
    child_info:CURRENT_CHILD_INFO;
  end record;

  end mtd_complex_types; – End of the package
```

```
--
-- ******************************************
-- Portable Ada Multitasking Analyzer System
--
--          Version 1.0
--
--  Designed, developed, and written by:
--          David O. LeVan
--          Robert J. Knapper
--             of the
-- Computer Software and Engineering Division
--    Institute for Defense Analyses
--          Alexandria, VA
--
--             11/8/88
--
-- ******************************************
--
with UNCHECKED_CONVERSION, UNCHECKED_DEALLOCATION;
package useful_generics is

  generic
   type T is private;
  package PDL_list_package is

    type T_list_block;
    type T_list_ptr is access T_list_block;
    type T_list_block is
--
-- Block to hold a data chunk and a pointer to the next in the list
--
      record
        DATA: T; -- The data itself
        LINK: T_list_ptr:= null; -- Pointer to next
      end record;

    type T_list_header is
--
-- Header for lists
--
      record
        FIRST,
           LAST,
           CURRENT: T_list_ptr:= null; -- Pointers
--
-- FIRST: Pointer to first element in the list
-- LAST: Pointer to the last element
```

```
-- CURRENT: Pointer to the next element, or current, to be read for sequential operations
--
    end record;

  type T_list_header_ptr is access T_list_header;-- Pointer for header block

  subtype END_OF_LIST is BOOLEAN;

      function list_length(
        L:in T_list_header) return INTEGER;
--
-- Function to return number of active elements in list
--
-- L: Pointer to header block for list to use
--
    procedure append(
        L: in out T_list_header;
        NewElement:T);
--
-- Appends a NewElement onto the end of the list pointed to by L
--
-- L: Pointer to list's header block
-- NewElement: New chunk of data to append
--
      procedure free is new UNCHECKED_DEALLOCATION(
        T_list_block,
        T_list_ptr);
--
-- Routine to free up the allocated memory.  Currently not implemented
--
      procedure read_and_consume(
        L: in out T_list_header;
        Data: out T;
        EOL: in out END_OF_LIST);
--
-- Reads one element, deletes it from the list.
--
-- L: Pointer to header
-- Data: Data element read
-- EOL: End of List set true if ran past end of list
--

    procedure read_nth_element(
        L: in out T_list_header;
      Data: out T;
        N: integer:= 1;
        EOL:in out END_OF_LIST);
```

```
--
-- Reads the Nth element on the list
--
-- L: Pointer to list header
-- Data: Receives data element just read
-- N: Number of element desired, numbered from 1 up
-- EOL: End of List encountered if true
--

        procedure update_nth_element(
          L: in out T_list_header;
          Data: T;
          N: INTEGER;
          EOL:in out END_OF_LIST);
--
-- Replaces current Nth element with Data element
--
-- L: as before
-- Data: data element to place on list at Nth position
-- N: Position in list to use
-- EOL: End of List
--

        procedure read_next_element(
          L: in out T_list_header;
          Data:out T;
          EOL:in out END_OF_LIST);
--
-- Performs sequential reads from the list, uses Current pointer in list header
--
-- L: Pointer to list header
-- Data: Data element read from list
-- EOL: Enf of List
--

        procedure reset_T_list(
          L: in out T_list_header);
--
-- Routine to reset pointer in header to first element
--
-- L: Pointer to header of list
--

        procedure consume(
          L: in out T_list_header;
          EOL: in out END_OF_LIST);
--
```

```
-- Routine to remove a list from allocated memory
--
-- L: List header pointer
-- EOL: End of List, or non-existent
--


  end PDL_list_package;


end useful_generics;
```

```
--
-- ****************************************
-- Portable Ada Multitasking Analyzer System
--
--          Version 1.0
--
--  Designed, developed, and written by:
--          David O. LeVan
--          Robert J. Knapper
--             of the
-- Computer Software and Engineering Division
--    Institute for Defense Analyses
--          Alexandria, VA
--
--             11/8/88
--
-- ****************************************
--
package body useful_generics is

  package body PDL_list_package is


   function list_length(L:in T_list_header) return INTEGER is
         N: INTEGER:=0;
         p:T_list_ptr:=L.first;

   begin
         if p=null then
            return N;
         end if;
-- count entries until hit the end

         loop
            N:=N+1;
            p:=p.link;
            exit when p=null;
         end loop;
         return N;
   end list_length;

      procedure append(
            L: in out T_list_header;
            NewElement:T) is
-- make new block
         new_list_block: T_list_ptr
               := new T_list_block'(LINK=>null, DATA=>NewElement);
```

```
begin
  if L.LAST = null then
    L.FIRST:= new_list_block; -- first time through
  else
    L.LAST.LINK:= new_list_block; -- normal action
  end if;
  L.LAST:= new_list_block;
end append;

   procedure read_and_consume(
     L: in out T_list_header;
     Data: out T;
     EOL: in out END_OF_LIST) is

begin
     EOL:=FALSE;
     if L.FIRST = null then
          EOL:= TRUE;
     else
       Data:= L.FIRST.DATA;
       consume(L,EOL);
          L.current:=L.first;
     end if;
end read_and_consume;

procedure read_nth_element(
     L: in out T_list_header;
  Data: out T;
     N:integer:= 1;
     EOL: in out END_OF_LIST) is

  i: integer:= N;
  p: T_list_ptr:= L.FIRST;

begin
     EOL:=FALSE;
     if i < 1 then
          EOL:= TRUE;
     else
-- find Nth element
       while i > 1 loop
         i:= i-1;
      p:= p.LINK;
         if p = null then
            EOL := TRUE;
            exit;
         end if;
```

```
            end loop;
            if not EOL then
           Data:= p.DATA;-- get the data

            end if;

            end if;

        end read_nth_element;

          procedure update_nth_element(
            L: in out T_list_header;
            Data: T;
            N: INTEGER;
            EOL:in out END_OF_LIST) is

                  p: T_list_ptr:= L.FIRST;
                  i:INTEGER:=N;

            begin
                  EOL:=FALSE;
                  if i < 1 then
                    EOL:= TRUE;
                  else
-- find the Nth element
                    while i > 1 loop
                    i:= i-1;
                    p:= p.LINK;
                    if p = null then
                          EOL:= TRUE;
                          exit;
                    end if;
                    end loop;
                    if not EOL then
                          p.DATA:=Data;-- update the element
                    end if;
                  end if;


            end update_nth_element;

          procedure read_next_element(
            L: in out T_list_header;
            Data:out T;
            EOL: in out END_OF_LIST) is

begin
```

```
  EOL:=FALSE;
  if L.current /= null then
       Data:= L.current.data; -- get the current element
       L.current:= L.current.link;-- move the pointer to the next
  else
       EOL:= TRUE;
  end if;
end read_next_element;

  procedure reset_T_list(L: in out T_list_header) is

begin
  L.current:=L.first;-- point to beginning
end reset_T_list;

        procedure consume(
          L: in out T_list_header;
          EOL:in out END_OF_LIST) is

          p,q: T_list_ptr;

     begin
          EOL:=FALSE;
          p:= L.FIRST;
          if p = null then
               EOL:= TRUE;
          end if;
          q:= p.LINK;
        -- free(p);
        L.FIRST:= q;
        if q = null then
             L.LAST:= q;
          end if;
     end consume;

  end PDL_list_package;

end useful_generics;
```

```
--
-- ******************************************
-- Portable Ada Multitasking Analyzer System
--
--          Version 1.0
--
--  Designed, developed, and written by:
--          David O. LeVan
--          Robert J. Knapper
--             of the
-- Computer Software and Engineering Division
--     Institute for Defense Analyses
--          Alexandria, VA
--
--             11/8/88
--
-- ******************************************
--
with mtd_complex_types;use mtd_complex_types;
with useful_types;use useful_types;
with mtd_fundamental_types;use mtd_fundamental_types;

package user_interface_types is
--


--
-- Data structures for controlling the report and breakpoint action
-- of the monitor.
--
type BREAK_PT is(
--
-- Indicates whether a breakpoint has been tripped
--
  tripped,
  reset);

type MTD_ACTION is(
--
-- This is the the manner the monitor is to use  a probe placed
-- onto the condition list.
--
  report, -- Just report, auto resume
  break,  -- Breakpoint, manual resume
  control); -- Probe exerts control over execution

type PROBE_CONDITION is(
--
```

```
-- What piece of probe information is to checked for a match
-- between the current probe and the probes on the condition list.
--
   check_name,
   check_id,
   check_module,
   check_task_action);

type PROBE_INFO
--
-- This is the basic unit of information kept about a probe on the
-- condition list.
--
is record
   module:MODULE_ID:=(-1,-1);
   name:STRING_REC;
   id:INTEGER:=-1;
   tasking_action:TASK_ACTION:=end_of_program;
   action:MTD_ACTION:=report;
   condition:PROBE_CONDITION:=check_id;
   active:BOOLEAN:=FALSE;
   break_status:BREAK_PT:=reset;
   control:STATE_CONTROL:=nop;
   control_info:STATE_CONTROL_INFO;
end record;

end user_interface_types;
```

```
--
-- ******************************************
-- Portable Ada Multitasking Analyzer System
--
--          Version 1.0
--
--  Designed, developed, and written by:
--          David O. LeVan
--          Robert J. Knapper
--             of the
-- Computer Software and Engineering Division
--     Institute for Defense Analyses
--          Alexandria, VA
--
--             11/8/88
--
-- ******************************************
--
-- This package defines routines for manipulating
-- varying-length character strings, as a_string (access string).

--

with useful_types; use useful_types;

package new_A_Strings is


        function a_str_length(s:STRING)
          return NATURAL; -- finds the length of an Ada string
--
-- Returns the length of an Ada string
--
-- s: a string
--


        function to_a(s:STRING)
          return STRING_REC;-- converts Ada string to record format
--
-- Converts an Ada string into a variable length string
--
-- s: Ada string
--

        function eq_string (S:STRING_REC; T:STRING_REC) return BOOLEAN;
--
-- Compares two variable length strings for equality
```

```
--
-- S: Variable length string
-- T: Another variable length string
--


end new_A_Strings;
```

```
--
-- ******************************************
-- Portable Ada Multitasking Analyzer System
--
--          Version 1.0
--
--  Designed, developed, and written by:
--          David O. LeVan
--          Robert J. Knapper
--             of the
-- Computer Software and Engineering Division
--     Institute for Defense Analyses
--          Alexandria, VA
--
--             11/8/88
--
-- ******************************************
--
-- This package defines types and routines for manipulating
-- varying-length character strings, as a_string (access string_rec).

-- SFZ 1/21/86
package body new_A_Strings is


        function a_str_length(s: STRING) return NATURAL is

        begin
          return NATURAL(s'LENGTH);
      end;

        function to_a(s:STRING) return STRING_REC is
      result: STRING_REC;
        begin
          result:= (s'LENGTH,s);
          return result;
        end to_a;

        function eq_string (S: STRING_REC; T:STRING_REC) return BOOLEAN is
        begin
          return s.s = t.s;
        end;
end new_A_Strings;
```

```
--
-- ****************************************
-- Portable Ada Multitasking Analyzer System
--
--          Version 1.0
--
--  Designed, developed, and written by:
--          David O. LeVan
--          Robert J. Knapper
--             of the
-- Computer Software and Engineering Division
--     Institute for Defense Analyses
--          Alexandria, VA
--
--             11/8/88
--
-- ****************************************
--
with useful_types; use useful_types;
with mtd_fundamental_types; use mtd_fundamental_types;
with mtd_complex_types; use mtd_complex_types;
with useful_generics; use useful_generics;
with new_a_strings; use new_a_strings;
with user_interface_types;use user_interface_types;

--
-- A module group is the base file, task, or generic package.  Each one is
-- a template for the cloning of others.
-- Therefore, each requires its own link
-- to the monitor task.  Each group may have many
-- instantiations of its base
-- template with each receiving its own number.  The unique combination of
-- a group plus this instantiation number forms the ID for modules.
-- When a link logs in it is assigned this module ID.
-- All probes using this
-- link must identify the module ID.  Each link may have many probes using it.
-- When a probe logs in it is assigned a uniques ID number by the monitor.
-- Which link is being used is recorded along with other
-- information.  An entry
-- in a list for each module is made recording
-- the fact that a particular probe
-- is using that link.  This is the link list.
-- : a GROUP is the basic template. May have many clones
-- : a MODULE is a particular instantiation of a group=> MODULE_ID
-- : a LINK LIST is the list of links (instantiations) of a group
-- : a LINK PROBE LIST is the list of probes using the link for a module
-- : a PROBE LIST is the list of all probes logged into the monitor.
```

```
--
-- Instantiate new list packages for each type of list in the
-- system
--
package data_control_support is
--
-- Spec. for routines to manipulate the monitor's internal data base
--
type MODULE_PROBE_ELEMENT
--
-- Element identifying a probe that is in the module
--
is record
  id:P_ID:=-1;
  alive:BOOLEAN:=FALSE;
end record;

package module_probes is new
  PDL_list_package(MODULE_PROBE_ELEMENT);
--
-- These functions define the equal operator for these pointer types
--

function "="(P1,P2:module_probes.T_list_ptr) return Boolean
        Renames module_probes."=";

function "="(P1,P2:module_probes.T_list_header_ptr) return Boolean
        Renames module_probes."=";
subtype MODULE_PROBES_HEADER_PTR is module_probes.T_LIST_HEADER_PTR;

type MODULE_ELEMENT
--
-- Element to contain module information on the group's list
--
is record
  dynamic: MODULE_INFO_D;
  alive: BOOLEAN:=FALSE;
  probe_list:MODULE_PROBES_HEADER_PTR:=null;
end record;


package modules is new
  PDL_list_package(MODULE_ELEMENT);


function "="(P1,P2:modules.T_list_header_ptr) return Boolean
        Renames modules."=";
```

```
subtype MODULES_HEADER_PTR is  modules.T_LIST_HEADER_PTR;

type MODULE_GROUP_ELEMENT
--
-- Group information element for group list
--
is record
   static: MODULE_INFO_S;
   group: G_ID;
   num_instans: NATURAL;
   module_list:MODULES_HEADER_PTR:=null;
end record;


package module_group is new
   PDL_list_package(module_group_element);

module_group_header: module_group.T_LIST_HEADER;

type ENTRY_STATS
--
-- Tracks task entry usage by other modules
--
is record
   parent:MODULE_ID;
   task_called_as: STRING_REC;
   entry_count: NATURAL;
end record;

package entry_usage is new
   PDL_list_package(ENTRY_STATS);

subtype ENTRY_STATS_HEADER_PTR is entry_usage.T_LIST_HEADER_PTR;

function "="(P1,P2:entry_usage.T_LIST_HEADER_PTR) return Boolean
         Renames entry_usage."=";

type PROBE_ELEMENT
--
-- All the information known about the probe for the master probe list.
--
is record
   static: PROBE_INFO_S;-- assigned at instrumentation time
   dynamic: PROBE_INFO_D;-- run-time info
   id:P_ID;-- ID assigned by monitor
   present_module: MODULE_ID;-- ID of module probe executes in
   alive:BOOLEAN:=FALSE;-- probe entry active
```

```
   total_entry_calls: NATURAL:=0;-- valid for probes after a task accept
   max_calls_in_Q: NATURAL:=0;-- maximum number of calls that were waiting on Q
   entry_stats_ptr: ENTRY_STATS_HEADER_PTR:= null;-- points to list
end record;

--
-- List to contain master probe information
--
package probes is new
   PDL_list_package(PROBE_ELEMENT);
probes_header: probes.T_LIST_HEADER;

--
-- Now the list to contain the probe information  record
--
package p_condition is new
   PDL_list_package(PROBE_INFO);


condition_header:p_condition.T_LIST_HEADER;

--
-- Types to specify the fields to search in the lists for  information
--
subtype END_OF_LIST is BOOLEAN;

type GROUP_FIELD is (
--
-- Data fields possible between the group's static and dynamic information
--
   file,
   module_name,
   module_type,
   modifier,
   group);

type MODULE_FIELD is (
--
-- The different fields of interest in modules
--
   id,
   action,
   alive);

type MODULE_PROBE_FIELD is (
--
-- Fields pertanent in module's list of probes
```

```
--
   id,
   alive);

type PROBE_FIELD is (
--
-- Match on the full probe description or just the specified one
--
   full,
   id,
   module,
   alive);

--
-- data structures for use in searching the lists
--
type SEARCH_GROUP(element:GROUP_FIELD:=group)
--
-- Type to handle data obtained through the group field search
--
is record
   case element is
   when file=>
         file_name:STRING_REC;
   when module_name=>
         module_n:STRING_REC;
   when module_type=>
         module_t:TYPE_OF_MODULE;
   when modifier=>
         modify:TYPE_OF_MODULE_MODIFIER;
   when group=>
         group_id: G_ID;
   end case;
end record;

type SEARCH_MODULE(element:MODULE_FIELD:=id)
--
-- Data type to handle the data obtained via a search using the module field
--
is record
   case element is
   when id=>
         m_id:MODULE_ID;
   when action=>
         act:LINK_ACTION;
   when alive=>
         live:BOOLEAN;
```

```
  end case;
end record;

type SEARCH_M_PROBE(element:MODULE_PROBE_FIELD:=id)
--
-- Type to contain the data obtained via a search for the module probe field
--
is record
  case element is
  when id=>
       probe_id:P_ID;
  when alive=>
       live:BOOLEAN;
  end case;
end record;

type SEARCH_PROBE(element:PROBE_FIELD:=id)
--
-- Type to hold data obtained via the probe field search
--
is record
  case element is
  when full=>
       full_entry:PROBE_ELEMENT;
  when id=>
       probe_id:P_ID;
  when module=>
       present_module:MODULE_ID;
  when alive=>
       live:BOOLEAN;
  end case;
end record;

type IDENT is (module,probe);-- identifies whether working with modules or probes

--
-- Functions and procedures to search the lists
--



procedure search_module_groups_for(
  L: in out module_group.T_LIST_HEADER;
  search_element: SEARCH_GROUP;
  result_element: in out MODULE_GROUP_ELEMENT;
  N:in out NATURAL;
  EOL: in out END_OF_LIST);
```

```
--
-- Searches the groups for a module matching the search group criterion
--
--
-- L: Pointer to the header of the list
-- search_element: element to search the list to find a module containing it.
-- result_element: Module info if a match is found
-- N: Location in list of the match.  Can be used by read Nth element procedures.
-- EOL: True if no match
--

procedure search_modules_for(
   L:in out modules.T_LIST_HEADER;
   search_element: SEARCH_MODULE;
   result_element:in out MODULE_ELEMENT;
   N:in out NATURAL;
   EOL: in out END_OF_LIST) ;
--
-- This searches a group's list of modules to find one satisfying the search field criterion
--
-- L: Pointer to header for the module list
-- search_element: Field in a module to find a match on
-- result_element: Contains matching module if a match is found
-- N: location in list of the matching module.
-- EOL: True if no match  is found
--

procedure search_module_probes(
   L:in out module_probes.T_LIST_HEADER;
   search_element: SEARCH_M_PROBE;
   result_element:in out MODULE_PROBE_ELEMENT;
   N:in out NATURAL;
   EOL: in out END_OF_LIST) ;
--
-- Searches the module's list of probes for a matching probe.
--
-- L:Pointer to list's header
-- search_element: Probe's field to search for
-- result_element: matching probe entry
-- N: Location in list of match
-- EOL: True if no match
--

procedure search_probes_for(
   L:in out probes.T_LIST_HEADER;
   search_element: SEARCH_PROBE;
   result_element:in out PROBE_ELEMENT;
```

```
   N:in out NATURAL;
   EOL:in out END_OF_LIST) ;
--
-- Searches the master probe list for a probe matching the criterion
--
-- L: Pointer to probe list
-- search_element: Probe's field to search for
-- result_element: Holds the matching probe's info
-- N: Location in list of the match
-- EOL: True if no match is found
--

procedure find_module(
   L:in out module_group.T_LIST_HEADER;
   search_element:MODULE_ID;
   result_group:in out MODULE_GROUP_ELEMENT;
   result_module:in out MODULE_ELEMENT;
   N_module:in out NATURAL;
   N_group:in out NATURAL;
   EOL:in out END_OF_LIST) ;
--
-- Given a module ID this routine finds out info about it, such as group and module data
--
-- L: Pointer to header of the list
-- search_element: Criterion to match
-- result_group: Matching group info
-- result_module: matching module info
-- N_Module: Location in module list of module entry
-- N_group: Location in group list of matching group info
-- EOL: True if no match found.
--

end data_control_support;
```

```
--
-- ******************************************
-- Portable Ada Multitasking Analyzer System
--
--          Version 1.0
--
--  Designed, developed, and written by:
--          David O. LeVan
--          Robert J. Knapper
--             of the
-- Computer Software and Engineering Division
--    Institute for Defense Analyses
--          Alexandria, VA
--
--             11/8/88
--
-- ******************************************
--
--
-- A module group is the base file, task, or generic package.  Each one is
-- a template for the cloning of others.
-- Therefore, each requires its own link
-- to the monitor task.  Each group may have many
-- instantiations of its base
-- template with each receiving its own number.  The unique combination of
-- a group plus this instantiation number forms the ID for modules.
-- When a link logs in it is assigned this module ID.
-- All probes using this
-- link must identify the module ID.  Each link may have many probes using it.
-- When a probe logs in it is assigned a uniques ID number by the monitor.
-- Which link is being used is recorded along with other
-- information.  An entry
-- in a list for each module is made recording
-- the fact that a particular probe
-- is using that link.  This is the link list.
-- : a GROUP is the basic template. May have many clones
-- : a MODULE is a particular instantiation of a group=> MODULE_ID
-- : a LINK LIST is the list of links (instantiations) of a group
-- : a LINK PROBE LIST is the list of probes using the link for a module
-- : a PROBE LIST is the list of all probes logged into the monitor.

--
-- Instantiate new list packages for each type of list in the
-- system
--
package body data_control_support is
--
```

```
-- Routines to manipulate the monitor's internal data base
--


--
-- Functions and procedures to search the lists
--



procedure search_module_groups_for(

   L: in out module_group.T_LIST_HEADER;
   search_element: SEARCH_GROUP;
   result_element: in out MODULE_GROUP_ELEMENT;
   N:in out NATURAL;
   EOL: in out END_OF_LIST) is

begin
   N:=0;
   module_group.reset_T_list(L);
   EOL:= FALSE;
   loop
         module_group.read_next_element(
           L,result_element,EOL);
         exit when EOL;
         N:=N+1;
         case search_element.element is
         when file =>
           exit when eq_string(
                 search_element.file_name,
                 result_element.static.file_name);-- compare two variable length strings

         when module_name =>
           exit when eq_string(
                 search_element.module_n,
                 result_element.static.module.module_name);

         when module_type =>
           exit when
                 search_element.module_t=
                 result_element.static.module.module_type;

         when modifier =>
           exit when
                 search_element.modify=
                 result_element.static.module.modifier;
```

```
        when group =>
          exit when
                search_element.group_id=result_element.group;
        end case;
  end loop;
end search_module_groups_for;

procedure search_modules_for(

  L:in out modules.T_LIST_HEADER;
  search_element: SEARCH_MODULE;
  result_element:in out MODULE_ELEMENT;
  N:in out NATURAL;
  EOL: in out END_OF_LIST) is

begin
  modules.reset_T_list(L);
  EOL:= FALSE;
  N:=0;

  loop
        modules.read_next_element(L,result_element,EOL);
        exit when EOL;
        N:=N+1;
        case search_element.element is
        when id =>
          exit when
                result_element.dynamic.id=
                search_element.m_id;

        when action =>
          exit when
                result_element.dynamic.action=
                search_element.act;

        when alive =>
          exit when
                result_element.alive=
             search_element.live;
        end case;
  end loop;
end search_modules_for;

procedure search_module_probes(
  L:in out module_probes.T_LIST_HEADER;
  search_element: SEARCH_M_PROBE;
  result_element:in out MODULE_PROBE_ELEMENT;
```

```
   N:in out NATURAL;
   EOL: in out END_OF_LIST) is

begin
   EOL:= FALSE;
   N:=0;
   module_probes.reset_T_list(L);
   loop
         module_probes.read_next_element(
           L,result_element,EOL);
         exit when EOL;
         N:=N+1;
         case search_element.element is
           when id=>
             exit when
                   search_element.probe_id=result_element.id;
           when alive=>
                 exit when
                   search_element.live=
                   result_element.alive;
         end case;
   end loop;
end search_module_probes;

procedure search_probes_for(

   L:in out probes.T_LIST_HEADER;
   search_element: SEARCH_PROBE;
   result_element:in out PROBE_ELEMENT;
   N:in out NATURAL;
   EOL:in out END_OF_LIST) is

begin
   N:=0;
   EOL:= FALSE;
   probes.reset_T_list(L);
   loop
         probes.read_next_element(
           L,result_element,EOL);
         exit when EOL;
         N:=N+1;
         case search_element.element is
         when full =>
           exit when
             search_element.full_entry=result_element;
         when id =>
           exit when
```

```
                result_element.id=search_element.probe_id;

        when module =>
          exit when
                result_element.present_module=
                search_element.present_module;
        when alive=>
          exit when
                result_element.alive=
                search_element.live;
        end case;
  end loop;
end search_probes_for;

procedure find_module(
  L:in out module_group.T_LIST_HEADER;
  search_element:MODULE_ID;
  result_group:in out MODULE_GROUP_ELEMENT;
  result_module:in out MODULE_ELEMENT;
  N_module:in out NATURAL;
  N_group:in out NATURAL;
  EOL:in out END_OF_LIST) is


begin

  search_module_groups_for(
        L,
        (group,search_element.g_num),
        result_group,
        N_group,EOL);
  if EOL then
-- we have a problem. A wild probe has logged in
        null;-- nop for now
  else
        search_modules_for(
          result_group.module_list.all,
          (id,search_element),
          result_module,
          N_module,EOL);
  end if;

end find_module;

end data_control_support;
```

```
--
-- ******************************************
-- Portable Ada Multitasking Analyzer System
--
--         Version 1.0
--
--  Designed, developed, and written by:
--         David O. LeVan
--         Robert J. Knapper
--           of the
-- Computer Software and Engineering Division
--    Institute for Defense Analyses
--         Alexandria, VA
--
--           11/8/88
--
-- ******************************************
--
with useful_types; use useful_types;
with mtd_fundamental_types; use mtd_fundamental_types;
with mtd_complex_types; use mtd_complex_types;
with user_interface_types; use user_interface_types;

--
-- A module group is the base file, task, or generic package.  Each one is
-- a template for the cloning of others.
-- Therefore, each requires its own link
-- to the monitor task.  Each group may have many
-- instantiations of its base
-- template with each receiving its own number.  The unique combination of
-- a group plus this instantiation number forms the ID for modules.
-- When a link logs in it is assigned this module ID.
-- All probes using this
-- link must identify the module ID.  Each link may have many probes using it.
-- When a probe logs in it is assigned a uniques ID number by the monitor.
-- Which link is being used is recorded along with other
-- information.  An entry
-- in a list for each module is made recording
-- the fact that a particular probe
-- is using that link.  This is the link list.
-- : a GROUP is the basic template. May have many clones
-- : a MODULE is a particular instantiation of a group=> MODULE_ID
-- : a LINK LIST is the list of links (instantiations) of a group
-- : a LINK PROBE LIST is the list of probes using the link for a module
-- : a PROBE LIST is the list of all probes logged into the monitor.
```

```
package mtd_data_control is
--
-- Spec. for routines to manipulate the monitor's internal data base
--

procedure create_module_entry(
   static: MODULE_INFO_S;
   dynamic: in out MODULE_INFO_D);
--
-- Makes an entry in the module list for a group.  If group does not exist, it is
-- created.
--
-- static: Static information for the module and group.  Assigned at instrumentation time.
-- dynamic: Info on this particular instantiation, receives unique ID assigned by monitor.
--

procedure create_probe_entry(
   name: STRING_REC;
   action: TASK_ACTION;
   path:STRING_REC;
   module: MODULE_ID;
   probe_ident: out P_ID);
--
-- Adds an entry for the probe on the list of all probes.  Also, adds entry to
-- module's list of probes.
--
-- name: Probe's name
-- action: Tasking action probe monitors
-- path: Scope of probe as measured from the definition of the link task
-- module: ID of the module the probe is active in.
-- probe_ident: ID number assigned by monitor at probe's login to the monitor.
--

procedure remove_probe_entry(
   ident: P_ID;
   EOL:in out BOOLEAN);
--
-- Removes the specified probe from the list of probes, and the modules list of probes
--
-- ident: Probe's ID
-- EOL: Error variable for non-existant probe
--

procedure remove_module_entry(
   ident: MODULE_ID;
   EOL:in out BOOLEAN);
--
```

```
-- Removes all information for a module from the group's list of modules.  In addition,
-- removes all probe information from module's list and master list of probes.
--
-- ident: ID of module (was assigned by monitor at module login point).
-- EOL: True if no such module on the lists
--

procedure get_probe_info(
   ident: P_ID;
   static: out PROBE_INFO_S;
   dynamic: out PROBE_INFO_D;
   module: out MODULE_ID;
   total_entry_calls,
   max_calls_in_Q:out NATURAL;
   EOL:in out BOOLEAN);
--
-- Returns detailed information about probe identified by ident.
--
-- ident: Probe's ID
-- static: Information assigned to probe at instrumentation time
-- dynamic: Probe's dynamic info, such as pointer to link task to use
-- module: ID of module probe is executing in
-- total_entry_calls: Valid for probes placed after a task accept.
-- max_calls_in_Q: Holds the largest number of waiting calls on the rendezvous queue
-- EOL: No such probe
--

procedure get_module_info(
   ident: MODULE_ID;
   static: out MODULE_INFO_S;
   dynamic: out MODULE_INFO_D;
   num_instantiations: out NATURAL;
   EOL:in out BOOLEAN);
--
-- Returns information fro module ident
--
-- ident: ID for module the information is desired about
-- static: Information assigned at instrumentation time
-- dynamic: Module's dynamic information
-- num_instantiations: How many modules of this type are currently active.  May be larger than one
-- for task types and generic packages.
-- EOL: No such module on lists

procedure get_group_info(
   ident:G_ID;
   static:out MODULE_INFO_S;
   num_instans:out NATURAL;
```

```
   EOL:in out BOOLEAN);
--
-- Get information for group ident
--
-- ident: ID for the group assigned at instrumentation time
-- static: Group's info assigned at instrumentation time
-- EOL: True if no such group logged in.
--



--
-- routines to set and get specific information regarding modules and probes
--

procedure set_probe_control(
   ident: P_ID;
   control: STATE_CONTROL;
   info: STATE_CONTROL_INFO;
   EOL:in out BOOLEAN);
--
-- Sets control information for probe ident.  Controls probe's actions, such as
-- simply continue with original code (NOP) or raise exception.
--
-- ident: Probe's ID number
-- control: Sets the state the probe is to enter
-- info: Auxillary control information needed for some of the states.
-- EOL: True if no such probe
--

procedure set_link_action(
   ident: MODULE_ID;
   action: LINK_ACTION;
   EOL:in out BOOLEAN);
--
-- Sets control information for the link for module ident
--
-- ident: ID of the link's module
-- action: What the link is to do, such as pass information on to probe and/or terminate.
-- EOL: True if no such module
--

procedure set_probe_wait(
   ident: P_ID;
   wait: BOOLEAN;
   EOL:in out BOOLEAN);
--
-- Sets the status of the flag indicating whether or not the probe is waiting
```

```
-- on the link task (paused).
--
-- ident: ID of probe
-- wait: Value to set the flag
-- EOL: True if no such probe
--

procedure set_parent_info(
   ident: P_ID;
   parent: MODULE_ID;
   call_name: STRING_REC;
   number_queued:NATURAL:=0;
   EOL:in out BOOLEAN);
--
-- Sets information for a task regarding the parent module that performed the task
-- call
--
-- ident: ID for the probe monitoring the accepts in a task
-- parent: Module ID for the module  making the task call
-- call_name: By what name this task entry was actually called
-- number_queued: How many calls are currently on the queue
-- EOL: Set true for non-existant probe or parent module
--


function probe_is_waiting(
   ident: P_ID) return BOOLEAN;
--
-- Test to see if the probe is waiting on the link
--
-- ident: Probe's ID
--

function probe_is_alive(
   ident: P_ID) return BOOLEAN;
--
-- Tests to see if the probe ident is assigned to an active probe
--
-- ident: Probe's ID
--

function module_is_alive(
   ident: MODULE_ID) return BOOLEAN;
--
-- Tests to see if the module ident is assigned to a currently active module
--
-- ident: ID for the module
```

```
--

function get_module_group(
  file_name: STRING_REC;
  module: LOGICAL_MODULE) return G_ID;
--
-- Returns information on the group module is a member of.  Used when module ID not known.
--
-- file_name: Source file containing the group. Group numbers must be unique.
-- module: Information that describes the module within the file.
--

function get_probe_id(
  module_group: G_ID;
  probe_name: STRING_REC) return P_ID;
--
-- Returns the probe's ID for the probe in module_group with the name of probe_name
--
-- module_group: group containing the probe
-- probe_name: Name assigned at instrumentation time.  Must be unique within group.
--


--
-- control routines for the data lists.
--

procedure reset_probe_list(length:out NATURAL);
--
-- Sets the current element to be the first.  Returns the number of active
-- elements on the list.
--
-- length: Number of active elements on the list
--


procedure read_next_probe(
  ident: out P_ID; -- -1 indicates at the end
  static: out PROBE_INFO_S;
  dynamic: out PROBE_INFO_D;
  module: out MODULE_ID);
--
-- Reads information for next active probe on master list.
--
-- ident: Next probe's ID.  Set to -1 if the list is read past EOL
-- static: Information for probe assigned at instrumentation time
-- dynamic: Probe's information at run time, such as pointer to link task to use.
-- module: Module ID that contains the probe
```

```
--

procedure reset_module_group_list(length:out NATURAL);
--
-- Sets the current element to be the first.  Returns the number of active
-- elements on the list.
--
-- length: Number of active elements on the list
--

procedure read_next_group(
   ident: out G_ID; -- -1 indicates at end
   static: out MODULE_INFO_S;
   num_instantiations: out NATURAL);
--
-- Sequentially reads information for the next group on the list.  Order of groups
-- is dependent upon task startup order.
--
-- ident: ID for next group.  Set to -1 if read past EOL occurs.
-- static: Information that was assigned at instrumentation time.
-- num_instantiations: How many active modules for this group.
--

procedure reset_link_list(
   ident: G_ID;
   length:out NATURAL;
   EOL:in out BOOLEAN);
--
-- Sets the current element to be the first.  Returns the number of active
-- elements on the list.
--
-- ident: ID for group list of modules.  Each module has its own link.
-- length: Number of active elements on the list
-- EOL: True if no such group
--


procedure read_next_link(
   ident: G_ID;
   dynamic: out MODULE_INFO_D;
   at_end: out BOOLEAN);
--
-- Sequentially reads link (module) information for the group
--
-- ident: Group ID to read module information for
-- dynamic: Module's information assigned at run time
-- at_end: True when the end of the list is encountered.
```

```
--

procedure reset_link_probe_list(
   ident: MODULE_ID;
   length:out NATURAL;
   EOL:in out BOOLEAN);
--
-- Sets the current element to be the first.  Returns the number of active
-- elements on the list.
--
-- ident: Module's ID for probe list.  Each module maintains a list of those probes in it.
-- length: Number of active elements on the list
-- EOL: True if no such module
--


procedure read_next_link_probe(
   ident: MODULE_ID;
   probe_id: out P_ID); -- -1 indicates at end
--
-- Sequentially reads information for the next probe in the module (using the same link)
--
-- ident: Module ID
-- probe_id: Probe's ID that is next on the list.
--


--
-- Probe statistics list basic operations
--

procedure reset_probe_stat_list(
   ident: P_ID;
   length: out NATURAL;
   EOL:in out BOOLEAN);
--
-- Sets the current element to be the first.  Returns the number of active
-- elements on the list.
--
-- ident: Probe's ID
-- length: Number of active elements on the list
-- EOL: True if no such probe
--


procedure read_next_probe_stat(
   ident:P_ID;
   parent:out MODULE_ID;
```

```
  called_as:out STRING_REC;
  entry_count:out NATURAL;
  EOL:in out BOOLEAN);
--
-- Sequentially reads next status entry for probe ident.
--
-- ident: ID for the probe we want the information from
-- parent: Module ID of the parent that made the call
-- called_as: The name by which the task call was made
-- entry_count: How many times this parent called this task
-- EOL: True if no such probe
--


--
-- Condition list operations
--

procedure reset_condition_list(
  length:out NATURAL);
--
-- Sets the current element to be the first.  Returns the number of active
-- elements on the list.
--
-- length: Number of active elements on the list
--


procedure read_next_condition(
  condition:in out PROBE_INFO;
  EOL:in out BOOLEAN);
--
-- Sequentially reads the next condition entry from the list
--
-- condition: Information for the probe and the conditions to monitor
-- EOL: True if at end of list
--

procedure read_nth_condition(
  num:in NATURAL;
  condition:in out PROBE_INFO;
  EOL:in out BOOLEAN);
--
-- Reads the Nth condition on the list
--
-- num: condition entry number
-- condition: Information for the probe
-- EOL: True if no such condition
```

```
--

procedure write_nth_condition(
   num:in NATURAL;
   condition:in PROBE_INFO;
   EOL:in out BOOLEAN);
--
-- Updates the Nth condition with the new condition information
--
-- num: Condition number to update
-- condition: Information to place on the list
-- EOL: True if no such condition number
--

procedure create_condition(
   condition:in PROBE_INFO);
--
-- Add the condition to the list.  Appends it only if all current entries are active.
--
-- condition: Information to be placed on the list
--

procedure remove_nth_condition(
   num:in NATURAL;
   action:in MTD_ACTION;
   EOL:in out BOOLEAN);
--
-- Sets the Nth condition as inactive.  This entry may then be reused by the create_condition.
--
-- num: Entry number
-- action: Which condition action this entry should be.  Error check.
-- EOL: True if no such entry or the action does not match
--

end mtd_data_control;
```

```
--
-- ****************************************
-- Portable Ada Multitasking Analyzer System
--
--          Version 1.0
--
--  Designed, developed, and written by:
--          David O. LeVan
--          Robert J. Knapper
--             of the
-- Computer Software and Engineering Division
--     Institute for Defense Analyses
--          Alexandria, VA
--
--             11/8/88
--
-- ****************************************
--
with text_io;use text_io;
with data_control_support;use data_control_support;
with new_a_strings; use new_a_strings;
--
-- A module group is the base file, task, or generic package.  Each one is
-- a template for the cloning of others.
-- Therefore, each requires its own link
-- to the monitor task.  Each group may have many
-- instantiations of its base
-- template with each receiving its own number.  The unique combination of
-- a group plus this instantiation number forms the ID for modules.
-- When a link logs in it is assigned this module ID.
-- All probes using this
-- link must identify the module ID.  Each link may have many probes using it.
-- When a probe logs in it is assigned a uniques ID number by the monitor.
-- Which link is being used is recorded along with other
-- information.  An entry
-- in a list for each module is made recording
-- the fact that a particular probe
-- is using that link.  This is the link list.
-- : a GROUP is the basic template. May have many clones
-- : a MODULE is a particular instantiation of a group=> MODULE_ID
-- : a LINK LIST is the list of links (instantiations) of a group
-- : a LINK PROBE LIST is the list of probes using the link for a module
-- : a PROBE LIST is the list of all probes logged into the monitor.

package body mtd_data_control is
--
-- Working variable definitions
```

```
--
result_group:MODULE_GROUP_ELEMENT;
result_module:MODULE_ELEMENT;
result_probe:PROBE_ELEMENT;
result_p_id:MODULE_PROBE_ELEMENT;
N_group,N_module,N_probe,N_p_id: INTEGER;
EOL:END_OF_LIST;

--
-- Procedures the outside world uses to manipulate the data lists
--

procedure create_module_entry(
  static: MODULE_INFO_S;
  dynamic: in out MODULE_INFO_D) is

begin
-- First find the group the module is assigned to

  search_module_groups_for(
       module_group_header,
    (group,dynamic.id.g_num),
       result_group,N_group,EOL);
-- If the group has not yet been logged in, do it

  if EOL then
       result_group.static:=static;
       result_group.group:=dynamic.id.g_num;
       result_group.num_instans:=0;
       result_group.module_list:=null;

    module_group.append(module_group_header,result_group);
       N_group:=N_group+1;
  end if;
-- now add the module to the group's list

  if result_group.module_list = null then
       result_group.module_list:= new modules.T_LIST_HEADER;-- used first time
       module_group.update_nth_element(module_group_header,
         result_group,N_group,EOL);
  end if;

-- look for an entry marked dead, conserves space
  search_modules_for(
       result_group.module_list.all,
       (alive,FALSE),
       result_module,
```

```
        N_module,EOL);

    if EOL then
      dynamic.id.l_num:= modules.list_length(
            result_group.module_list.all)+1;-- ID is length of list + one

      result_module.dynamic:=dynamic;
         result_module.alive:=TRUE;
         result_module.probe_list:=null;

      modules.append(result_group.module_list.all,result_module);

    else
      dynamic.id.l_num:=N_module;-- ID is position in list

      result_module.dynamic:=dynamic;
         result_module.alive:=TRUE;
         result_module.probe_list:=null;

         modules.update_nth_element(
           result_group.module_list.all,
           result_module,N_module,EOL);
    end if;
--
-- Now increment the number of instantiations entry
--
    result_group.num_instans:=result_group.num_instans+1;
    module_group.update_nth_element(module_group_header,
            result_group,N_group,EOL);

end create_module_entry;

procedure create_probe_entry(
    name: STRING_REC;
    action: TASK_ACTION;
    path:STRING_REC;
    module: MODULE_ID;
    probe_ident: out P_ID) is

l_id:P_ID;
result_probe:PROBE_ELEMENT;-- start fresh every time

begin
-- see if there is a dead probe in the list, will reuse it
    search_probes_for(
          probes_header,
          (alive,FALSE),
```

```
        result_probe,N_probe,EOL);
    result_probe.alive:=TRUE;
    result_probe.static:=((name.len,name.s),action,(path.len,path.s));
    result_probe.dynamic:=(nop,(1,DURATION(1)),FALSE,(-1,-1),(1," "));
    result_probe.id:= -1;
    result_probe.present_module:=module;
    result_probe.total_entry_calls:=0;
    result_probe.max_calls_in_Q:=0;
    result_probe.entry_stats_ptr:=null;

    if EOL then
-- no entry to reuse
      l_id:=probes.list_length(probes_header)+1;
        probe_ident:=l_id;
        result_probe.id:=l_id;
      probes.append(probes_header,result_probe);
    else
-- reuse the entry
        probe_ident:=N_probe;
        result_probe.id:=N_probe;
        probes.update_nth_element(probes_header,result_probe,
          N_probe,EOL);
    end if;



--
-- now update list of probes for the module the probe is in, we know only module ID here
--
    find_module(module_group_header,
        module,
        result_group,
        result_module,
        N_module,N_group,EOL);-- get other info on module

        if EOL then
-- the module has not logged in before the probe, do nothing
          null;
        else
          if result_module.probe_list=null then
              result_module.probe_list:=new module_probes.T_LIST_HEADER;-- first time
              modules.update_nth_element(
                result_group.module_list.all,
                result_module,N_module,EOL);
          end if;

          search_module_probes(
              result_module.probe_list.all,
```

```
            (alive,FALSE),
            result_p_id,
            N_p_id,EOL);-- find any dead entry

        result_p_id:=(result_probe.id,TRUE);
        if EOL then
        module_probes.append(
            result_module.probe_list.all,
            result_p_id);
        else
            module_probes.update_nth_element(
              result_module.probe_list.all,
              result_p_id,N_p_id,EOL);
        end if;
      end if;



end create_probe_entry;

procedure remove_probe_entry(
  ident: P_ID;EOL:in out BOOLEAN) is

begin
  EOL:=FALSE;
  probes.read_nth_element(probes_header,
      result_probe,ident,EOL);
  if not EOL then
  result_probe.alive:=FALSE;
  probes.update_nth_element(probes_header,
      result_probe,ident,EOL);-- mark it dead
-- now find the module the probe was in and remove
-- the probe from the module's list

  find_module(module_group_header,
      result_probe.present_module,
      result_group,result_module,
      N_module,N_group,EOL);

  if EOL then
-- No such module!!
      null;
  else
      search_module_probes(
        result_module.probe_list.all,
        (id,result_probe.id),
        result_p_id,N_p_id,EOL);
```

```
        if not EOL then
          result_p_id.alive:=FALSE;
          module_probes.update_nth_element(
                result_module.probe_list.all,
                result_p_id,N_p_id,EOL);-- mark it dead
        end if;
    end if;
end if;
end remove_probe_entry;

procedure remove_module_entry(
  ident: MODULE_ID;EOL:in out BOOLEAN) is

condition_info:PROBE_INFO;

begin
  EOL:=FALSE;
  find_module(module_group_header,
        ident,result_group,result_module,
        N_module,N_group,EOL);
  if EOL then
-- no such module
        null;
  else
        result_module.alive:=FALSE;
        modules.update_nth_element(
          result_group.module_list.all,
          result_module,N_module,EOL);-- mark it dead
--
-- Now note that one module has been removed
--
        result_group.num_instans:=result_group.num_instans-1;
        module_group.update_nth_element(
          module_group_header,result_group,N_group,EOL);

-- now prepare to kill the probe list for the module

        for i in 1..module_probes.list_length(
                result_module.probe_list.all) loop

          module_probes.read_nth_element(
                result_module.probe_list.all,
                result_p_id,i,EOL);
          result_p_id.alive:=FALSE;
          module_probes.update_nth_element(
                result_module.probe_list.all,
                result_p_id,i,EOL); -- mark it dead in the modules list
```

```
            probes.read_nth_element(
                  probes_header,result_probe,
                  result_p_id.id,EOL);
            result_probe.alive:=FALSE;
            probes.update_nth_element(
                  probes_header,result_probe,
                  result_p_id.id,EOL);-- mark it dead in master probe list
          end loop;

--
-- Search condition list for all probes belonging to the module
-- just removed.  When found, deactivate them.
--
          for i in 1..p_condition.list_length(condition_header) loop

            p_condition.read_nth_element(condition_header,
                  condition_info,i,EOL);
            if condition_info.active then
                  if ((condition_info.condition=check_id) or
                    (condition_info.condition=check_module)) and
                    (condition_info.module=result_module.dynamic.id) then
                  condition_info.active:=FALSE;
                  p_condition.update_nth_element(condition_header,
                    condition_info,i,EOL);
                  end if;
            end if;-- active?
          end loop;-- condition list

    end if;-- if module exists

end remove_module_entry;

procedure get_probe_info(
    ident: P_ID;
    static: out PROBE_INFO_S;
    dynamic: out PROBE_INFO_D;
    module: out MODULE_ID;
    total_entry_calls,max_calls_in_Q:out NATURAL;EOL:in out BOOLEAN) is

begin
    EOL:=FALSE;
    probes.read_nth_element(probes_header,
          result_probe,ident,EOL);-- ident is also index
    if not EOL and result_probe.alive then
      static:=result_probe.static;
          dynamic:=result_probe.dynamic;
      module:=result_probe.present_module;
```

```
        total_entry_calls:=result_probe.total_entry_calls;
        max_calls_in_Q:=result_probe.max_calls_in_Q;
    else
        EOL:=TRUE;
    end if;
end get_probe_info;


procedure get_module_info(
    ident: MODULE_ID;
    static: out MODULE_INFO_S;
    dynamic: out MODULE_INFO_D;
    num_instantiations: out NATURAL;EOL:in out BOOLEAN) is

begin
    EOL:=FALSE;
    find_module(module_group_header,
        ident,result_group,result_module,
        N_module,N_group,EOL);
    if not EOL and result_module.alive then
        static:=result_group.static;
        dynamic:=result_module.dynamic;
        num_instantiations:= result_group.num_instans;
    else
        EOL:=TRUE;
    end if;
end get_module_info;

procedure get_group_info(
    ident:G_ID;
    static:out MODULE_INFO_S;
    num_instans:out NATURAL;EOL:in out BOOLEAN) is

begin

    EOL:=FALSE;
    search_module_groups_for(
        module_group_header,
      (group,ident),
        result_group,N_group,EOL);
    if not EOL then
    static:=result_group.static;
    num_instans:=result_group.num_instans;
    end if;
end get_group_info;
```

```
--
-- routines to set and get specific information regarding modules and probes
--

procedure set_probe_control(
  ident: P_ID;
  control: STATE_CONTROL;
  info: STATE_CONTROL_INFO;EOL:in out BOOLEAN) is

begin
  EOL:= FALSE;
  probes.read_nth_element(probes_header,
       result_probe,ident,EOL);
  if EOL then
-- no such probe logged in
       null;
  else
       result_probe.dynamic.action:=control;
       result_probe.dynamic.control_info:=info;
       probes.update_nth_element(probes_header,
         result_probe,ident,EOL);
  end if;

end set_probe_control;

procedure set_link_action(
  ident: MODULE_ID;
  action: LINK_ACTION;EOL:in out BOOLEAN) is

begin
  EOL:=FALSE;
  find_module(module_group_header,ident,
       result_group,result_module,
       N_module,N_group,EOL);
  if EOL then
-- no such module
       null;
  else
    result_module.dynamic.action:=action;
       modules.update_nth_element(result_group.module_list.all,
         result_module,N_module,EOL);
  end if;

end set_link_action;

procedure set_probe_wait(
  ident: P_ID;
```

```
  wait: BOOLEAN;EOL:in out BOOLEAN) is

begin
  EOL:=FALSE;
  probes.read_nth_element(probes_header,
       result_probe,ident,EOL);
  if EOL then
-- no such probe
       null;
  else
       result_probe.dynamic.waiting:=wait;
       probes.update_nth_element(probes_header,
         result_probe,ident,EOL);
  end if;

end set_probe_wait;

procedure set_parent_info(
  ident: P_ID;
  parent: MODULE_ID;
  call_name: STRING_REC;
  number_queued:NATURAL:=0;EOL:in out BOOLEAN) is

entry_info:ENTRY_STATS;
found: BOOLEAN:=FALSE;

begin
  EOL:=FALSE;
  probes.read_nth_element(probes_header,
       result_probe,ident,EOL);
  if EOL then
-- no such probe
       null;
  else
       result_probe.dynamic.parent:=parent;
       result_probe.dynamic.task_call_name:=call_name;

       case result_probe.static.action is

       when start_rendezvous =>
-- must update entry usage list

         if result_probe.entry_stats_ptr = null then
              result_probe.entry_stats_ptr:=new entry_usage.T_LIST_HEADER;-- first time
         end if;
         found:= FALSE;
         for i in 1..entry_usage.list_length(
```

```
                result_probe.entry_stats_ptr.all) loop
                entry_usage.read_nth_element(
                  result_probe.entry_stats_ptr.all,entry_info,i,EOL); -- get the next entry
                if (entry_info.parent=parent) and
                  eq_string(entry_info.task_called_as, call_name) then
-- we have a match, this parent has called before-> update usage
                  found:=TRUE;
                  if entry_info.entry_count < NATURAL'LAST then
                        entry_info.entry_count:=entry_info.entry_count+1;
                        entry_usage.update_nth_element(
                          result_probe.entry_stats_ptr.all,entry_info,i,EOL);
                  end if;
                end if;-- found parent
          end loop;-- entry stats list

          if not found then
-- we have a new caller

                entry_info.parent:=parent;
                entry_info.task_called_as:= call_name;
                entry_info.entry_count:=1;
                entry_usage.append(result_probe.entry_stats_ptr.all,entry_info);
          end if;

          if result_probe.total_entry_calls<NATURAL'LAST then
                result_probe.total_entry_calls:=result_probe.total_entry_calls+1;
          end if;

          if number_queued>result_probe.max_calls_in_Q then
                result_probe.max_calls_in_Q:=number_queued;
          end if;
--
-- Now indicate parent is waiting while child executes starting at probe
--
          find_module(module_group_header,
                result_probe.dynamic.parent,
                result_group,result_module,
                N_module,N_group,EOL);
          if EOL then
-- No parent match
                null;
          else
                result_module.dynamic.child_info.is_executing:=TRUE;
                result_module.dynamic.child_info.at_probe:=ident;
                modules.update_nth_element(
                  result_group.module_list.all,result_module,
                  N_module,EOL);
```

```
            end if;

--
-- Now indicate parent is not waiting while child executes
--
          when end_accept | task_terminate | task_abort =>

            find_module(module_group_header,
                  result_probe.dynamic.parent,
                  result_group,result_module,
                  N_module,N_group,EOL);
            if EOL then
-- No parent match
                  null;
            else
                  result_module.dynamic.child_info.is_executing:=FALSE;

                  modules.update_nth_element(
                    result_group.module_list.all,result_module,
                    N_module,EOL);
            end if;

          when others=>

            null;

          end case;

          probes.update_nth_element(probes_header,
            result_probe,ident,EOL);

    end if;-- for EOL of probe list, rogue probe!!


end set_parent_info;

function probe_is_waiting(
  ident: P_ID) return BOOLEAN is


begin

  probes.read_nth_element(probes_header,
        result_probe,ident,EOL);
  if EOL or not result_probe.alive then
-- no such probe
        return FALSE;
```

```
  else
        return result_probe.dynamic.waiting;
  end if;

end probe_is_waiting;


function probe_is_alive(
  ident: P_ID) return BOOLEAN is
begin
  probes.read_nth_element(probes_header,
        result_probe,ident,EOL);
  if EOL then
-- no such probe
        null;
  else
        return result_probe.alive;
  end if;
end probe_is_alive;

function module_is_alive(
  ident: MODULE_ID) return BOOLEAN is
begin
  find_module(module_group_header,
        ident,result_group,
        result_module,N_module,N_group,EOL);
  return result_module.alive;
end module_is_alive;


function get_module_group(
  file_name: STRING_REC;
  module: LOGICAL_MODULE) return G_ID is


begin

  module_group.reset_T_list(module_group_header);
  loop
        module_group.read_next_element(module_group_header,
          result_group,EOL);
        exit when EOL;
        if eq_string(file_name,result_group.static.file_name) and
          (result_group.static.module=module) then
          return result_group.group;
        end if;
  end loop;
```

```
  return -1;     -- no match found

end get_module_group;

function get_probe_id(
  module_group: G_ID;
  probe_name: STRING_REC) return P_ID is

N:P_ID:=0;
found:boolean:=FALSE;

begin

for i in 1..probes.list_length(probes_header) loop
  probes.read_nth_element(probes_header,
       result_probe,i,EOL);
  N:=i;

  found:=eq_string(probe_name,result_probe.static.probe_name) and
       (result_probe.present_module.g_num = module_group) and
       result_probe.alive;
  exit when found;

end loop;
if found then
  return N;
else
  return -1;
end if;

end get_probe_id;

--
-- control routines for the data lists.
--

procedure reset_probe_list(length:out NATURAL) is

begin
  probes.reset_T_list(probes_header);
  length:=probes.list_length(probes_header);
end reset_probe_list;

procedure read_next_probe(
  ident: out P_ID; -- -1 indicates at the end
  static: out PROBE_INFO_S;
  dynamic: out PROBE_INFO_D;
```

```
  module: out MODULE_ID) is

begin
loop
  probes.read_next_element(probes_header,
       result_probe,EOL);
  exit when EOL or result_probe.alive;
end loop;

  if not EOL then
       ident:=result_probe.id;
       static:=result_probe.static;
       dynamic:=result_probe.dynamic;
       module:=result_probe.present_module;
  else
       ident:=-1;
  end if;

end read_next_probe;

procedure reset_module_group_list(length:out NATURAL) is

begin
  module_group.reset_T_list(module_group_header);
  length:=module_group.list_length(module_group_header);
end reset_module_group_list;

procedure read_next_group(
  ident: out G_ID; --1 indicates at end
  static: out MODULE_INFO_S;
  num_instantiations: out NATURAL) is

begin
  module_group.read_next_element(
       module_group_header,
       result_group,EOL);
  if not EOL then
       ident:=result_group.group;
       static:= result_group.static;
       num_instantiations:= result_group.num_instans;
  else
       ident:=-1;
  end if;

end read_next_group;

procedure reset_link_list(ident: G_ID; length:out NATURAL;
```

```
  EOL:in out BOOLEAN) is

begin
  EOL:=FALSE;
  search_module_groups_for(
        module_group_header,
        (group,ident),
        result_group,N_group,EOL);
  if EOL then
-- no such group
        null;
  else
        modules.reset_T_list(result_group.module_list.all);
        length:=modules.list_length(result_group.module_list.all);
  end if;

end reset_link_list;

procedure read_next_link(
  ident: G_ID;
  dynamic: out MODULE_INFO_D;
  at_end: out BOOLEAN) is

begin

  search_module_groups_for(
        module_group_header,
        (group,ident),
        result_group,N_group,EOL);
  if EOL then
-- no such group
        null;
  else
        loop
          modules.read_next_element(
                result_group.module_list.all,
                result_module,
                EOL);
          exit when EOL or result_module.alive;
        end loop;

        if not EOL then
          dynamic:=result_module.dynamic;
        end if;
  end if;
  at_end:=EOL;
```

```
end read_next_link;

procedure reset_link_probe_list(ident: MODULE_ID; length:out NATURAL;
  EOL:in out BOOLEAN) is

begin
  EOL:=FALSE;
  find_module(module_group_header,
       ident,result_group,result_module,
       N_module,N_group,EOL);
  if EOL then
-- no such module
       null;
  else
       module_probes.reset_T_list(result_module.probe_list.all);
       length:=module_probes.list_length(result_module.probe_list.all);
  end if;

end reset_link_probe_list;

procedure read_next_link_probe(
  ident: MODULE_ID;
  probe_id: out P_ID) is -- -1 indicates at end

begin

  find_module(module_group_header,
       ident,result_group,result_module,
       N_module,N_group,EOL);
  if EOL then
-- no such module
       probe_id:=-1;
  else
       loop
         module_probes.read_next_element(
              result_module.probe_list.all,
              result_p_id,EOL);
         exit when EOL or result_p_id.alive;
       end loop;

       if not EOL then
         probe_id:=result_p_id.id;
       else
         probe_id:=-1;
       end if;
  end if;
```

```
end read_next_link_probe;

--
-- Probe statistics list basic operations
--

procedure reset_probe_stat_list(
  ident: P_ID; length: out NATURAL; EOL:in out BOOLEAN) is

begin
  probes.read_nth_element(probes_header,result_probe,ident,EOL);
  if not EOL then
       if result_probe.entry_stats_ptr=null then
         EOL:=TRUE;
       else
         entry_usage.reset_T_list(
              result_probe.entry_stats_ptr.all);
         length:=entry_usage.list_length(
              result_probe.entry_stats_ptr.all);
       end if;
  end if;
end reset_probe_stat_list;

procedure read_next_probe_stat(
  ident:P_ID;
  parent:out MODULE_ID;
  called_as:out STRING_REC;
  entry_count:out NATURAL;
  EOL:in out BOOLEAN) is

entry_info:ENTRY_STATS;

begin
  probes.read_nth_element(probes_header,
       result_probe,ident,EOL);
  if not EOL then
       if result_probe.entry_stats_ptr=null or
         not result_probe.alive then
         EOL:=TRUE;
       else
         entry_usage.read_next_element(
              result_probe.entry_stats_ptr.all,entry_info,EOL);
         if not EOL then
              parent:= entry_info.parent;
              called_as:= entry_info.task_called_as;
              entry_count:= entry_info.entry_count;
         end if;
```

```
        end if;
    end if;
end read_next_probe_stat;


--
-- Condition list operations
--

procedure reset_condition_list(
    length:out NATURAL) is

begin

    p_condition.reset_T_list(condition_header);
    length:=p_condition.list_length(condition_header);
end reset_condition_list;

procedure read_next_condition(
    condition:in out PROBE_INFO;
    EOL:in out BOOLEAN) is


begin
    p_condition.read_next_element(condition_header,
        condition,EOL);
end read_next_condition;

procedure read_nth_condition(
    num:in NATURAL;
    condition:in out PROBE_INFO;
    EOL:in out BOOLEAN) is

begin

    p_condition.read_nth_element(condition_header,
        condition,num,EOL);

end read_nth_condition;

procedure write_nth_condition(
    num:in NATURAL;
    condition:in PROBE_INFO;
    EOL:in out BOOLEAN) is

begin
```

```
  p_condition.update_nth_element(condition_header,
       condition,num,EOL);
end write_nth_condition;


procedure create_condition(
  condition:in PROBE_INFO) is

found: BOOLEAN;
t_condition_info: PROBE_INFO;
EOL: BOOLEAN:=FALSE;
id: NATURAL;

begin

          found:=FALSE;
          for i in 1..p_condition.list_length(condition_header) loop
               p_condition.read_nth_element(condition_header,
                 t_condition_info,i,EOL);
               if not t_condition_info.active then
                 found:=TRUE;
                 id:=i;
                 exit;
               end if;
          end loop;
          if found then
                  p_condition.update_nth_element(condition_header,
                       condition,id,EOL);
          else
                  p_condition.append(condition_header,condition);
          end if;
end create_condition;

procedure remove_nth_condition(
  num:in NATURAL;
  action:in MTD_ACTION;
  EOL:in out BOOLEAN) is

condition_info:PROBE_INFO;

begin

        p_condition.read_nth_element(condition_header,condition_info,num,EOL);

        if EOL then
          put_line("No such condition in list!");
        elsif not condition_info.active then
```

```
            put_line("Not an active list entry!");
        else
                if (condition_info.action=action) then
                condition_info.active:=FALSE;
                p_condition.update_nth_element(condition_header,
                  condition_info,num,EOL);
                else
                put_line("Not correct entry type!");
                end if;
        end if;
end remove_nth_condition;


end mtd_data_control;
```

```
--
-- ******************************************
-- Portable Ada Multitasking Analyzer System
--
--          Version 1.0
--
--  Designed, developed, and written by:
--          David O. LeVan
--          Robert J. Knapper
--             of the
-- Computer Software and Engineering Division
--    Institute for Defense Analyses
--          Alexandria, VA
--
--             11/8/88
--
-- ******************************************
--
with mtd_fundamental_types;use mtd_fundamental_types;
with mtd_complex_types;use mtd_complex_types;

package user_interface_support is

   procedure display_groups;
--
-- Displays to user the list of groups.
--

   procedure display_modules(
         group:G_ID;
         EOL:in out BOOLEAN);
--
-- Displays to user the modules in group.
--
-- group: Group interested in
-- EOL: True if no such group
--

   procedure display_module_probes(
         module:MODULE_ID;
         EOL:in out BOOLEAN);
--
-- Displays the list of probes for module.
--
-- module: ID of the module with the list.
-- EOL: True if no such module
--
```

```
   procedure display_all_probes;
--
-- Displays all active probes in the system
--

   procedure display_waiting_probes;
--
-- Displays to user only those probes that are waiting on a link.  The enclosing modules
-- are in a paused condition.
--


   procedure display_group_details(
         group:G_ID;
         EOL:in out BOOLEAN);
--
-- Displays details of group.
--
-- group: ID of group the details are wanted for.
-- EOL: True if no such group
--

   procedure display_probe_details(
         probe:P_ID;
         EOL:in out BOOLEAN);
--
-- Displays to user the details of probe
--
-- probe: ID of probe to display
-- EOL: True if no such probe
--

   procedure display_module_details(
         module:MODULE_ID;
         EOL:in out BOOLEAN);
--
-- Displays the details to the user for module
--
-- module: ID of module the details are wanted for.
-- EOL: True if no such module
--

   procedure display_probe_stats(
         probe:P_ID;
         EOL:in out BOOLEAN);
--
-- Displays to user the list of entry usage statistics for the probe.  Valid only
```

```
-- for those probes placed after an accept.
--
-- probe: Probe ID
-- EOL: Nonexistant probe or no list for this probe.
--

end user_interface_support;
```

```
--
-- ****************************************
-- Portable Ada Multitasking Analyzer System
--
--          Version 1.0
--
--  Designed, developed, and written by:
--          David O. LeVan
--          Robert J. Knapper
--             of the
-- Computer Software and Engineering Division
--    Institute for Defense Analyses
--          Alexandria, VA
--
--             11/8/88
--
-- ****************************************
--
with text_io; use text_io;
with mtd_data_control;use mtd_data_control;
with useful_types;use useful_types;

package body user_interface_support is

package int_io is new integer_io(INTEGER);use int_io;
package state_control_enum is new text_io.enumeration_io(STATE_CONTROL);
use state_control_enum;
package task_act_enum is new text_io.enumeration_io(TASK_ACTION);
use task_act_enum;
package link_action_enum is new text_io.enumeration_io(LINK_ACTION);
use link_action_enum;

package type_module_enum is new text_io.enumeration_io(TYPE_OF_MODULE);
use type_module_enum;
package module_modifier_enum is new text_io.enumeration_io(TYPE_OF_MODULE_MODIFIER);
use module_modifier_enum;
package boolean_enum is new enumeration_io(BOOLEAN);
use boolean_enum;
package fix_io is new fixed_io(DURATION);use fix_io;

p_ident:constant STRING:="Probe ID:";
p_name:constant STRING:="Probe Name:";
t_action:constant STRING:="Task Action:";
p_path:constant STRING:="Probe Path:";
p_action:constant STRING:="Probe Action";
except_sel:constant STRING:="Select Exception:";
d_val:constant STRING:="Delay Value:";
```

```
wait:constant STRING:="Probe Waiting:";
parent:constant STRING:="Parent Module ID:";
called_as:constant STRING:="Task Called As:";
present:constant STRING:="Present Module ID:";

g_ident:constant STRING:="Group ID:";
f_name:constant STRING:="Module File:";
n_instans:constant STRING:="Number of instantiations:";
m_name:constant STRING:="Module Name:";
m_type:constant STRING:="Module Type:";
m_mod:constant STRING:="Module Modifier:";

l_action:constant STRING:="Link Action:";

line_length:constant INTEGER:=128;
line_buffer:STRING(1..line_length);
group_field,mod_field:STRING(1..4);


   probe_static: PROBE_INFO_S;
   probe_dynamic: PROBE_INFO_D;
   module_static: MODULE_INFO_S;
   module_dynamic: MODULE_INFO_D;
   present_module: MODULE_ID;
   present_probe: P_ID;
   module_group: G_ID;
   num_of_instantiations: NATURAL;
   at_eol: CONSTANT INTEGER:= -1;
   present_probe_name, present_file_name: STRING_REC;
   present_module_name: LOGICAL_MODULE;
   shut_down_all: BOOLEAN:= FALSE;
   EOL: BOOLEAN;
   length:NATURAL;
   total_entry_calls,max_calls_in_Q:NATURAL:=0;

procedure display_probe_details(probe:P_ID;EOL:in out BOOLEAN) is

begin

   EOL:=FALSE;

   get_probe_info(probe,probe_static,
         probe_dynamic,present_module,
         total_entry_calls,max_calls_in_Q,EOL);

   if not EOL and probe_is_alive(probe) then
         new_line;
```

```
        put("Probe Details:");new_line;

        put(p_ident);put(probe);new_line;

        put(p_name);put_line(probe_static.probe_name.s);

        put(t_action);put(probe_static.action);new_line;

        put(p_path);put(probe_static.probe_path.s);new_line;

        put(p_action);put(probe_dynamic.action);new_line;

        put(except_sel);put(probe_dynamic.control_info.sel_except);
        new_line;

        put(d_val);put(probe_dynamic.control_info.delay_val);
        new_line;

        put(wait);put(probe_dynamic.waiting);new_line;

        put(parent);put(probe_dynamic.parent.g_num);
        put(" , ");put(probe_dynamic.parent.l_num);new_line;

        put(called_as);put_line(probe_dynamic.task_call_name.s);

        put(present);put(present_module.g_num);
        put(" , ");put(present_module.l_num);new_line;
    end if;

end display_probe_details;



    procedure display_groups is

begin

    reset_module_group_list(length);

    put("List of groups.");new_line(2);

    loop
        read_next_group(module_group,
          module_static,
          num_of_instantiations);
        exit when module_group=at_eol;
```

```
        display_group_details(module_group,EOL);
    end loop;

end display_groups;

    procedure display_modules(group:G_ID;EOL:in out BOOLEAN) is

begin
    EOL:=FALSE;
    reset_link_list(group,length,EOL);
    if not EOL then
        put("List of modules for group:");put(group);new_line(2);

        for i in 1..length loop

            display_module_details(
                (group,i),EOL);

        end loop;
    end if;
end display_modules;

    procedure display_module_probes(module:MODULE_ID;EOL:in out BOOLEAN) is
begin
    EOL:=FALSE;
    reset_link_probe_list(module,length,EOL);

    if not EOL then
        put("List of probes for module:");put(module.g_num);
        put(" , ");put(module.l_num);new_line(2);

        loop
            read_next_link_probe(module,
                present_probe);
            exit when present_probe=at_eol;
            get_probe_info(present_probe,probe_static,
                probe_dynamic,present_module,
                total_entry_calls,max_calls_in_Q,EOL);
            put(p_ident);put(present_probe);put("  ");
                put(probe_static.probe_name.s);new_line;
        end loop;

    end if;
end display_module_probes;

    procedure display_all_probes is
EOL:BOOLEAN:=FALSE;
```

```
begin

   reset_probe_list(length);
   for i in 1..length loop
         if probe_is_alive(i) then
            display_probe_details(i,EOL);
         end if;
   end loop;

end display_all_probes;

   procedure display_waiting_probes is
EOL:BOOLEAN:=FALSE;
begin

   reset_probe_list(length);
   for i in 1..length loop
         if probe_is_waiting(i) then
            display_probe_details(i,EOL);
         end if;
   end loop;

end display_waiting_probes;

   procedure display_group_details(group:G_ID;EOL:in out BOOLEAN) is
begin
   EOL:=FALSE;
   get_group_info(group,module_static,
         num_of_instantiations,EOL);
   if not EOL then
         new_line;
         put("Group Details:");new_line(2);

         put(g_ident);put(group);new_line;

         put(f_name);put_line(module_static.file_name.s);

         put(m_name);put_line(module_static.module.module_name.s);

         put(m_type);put(module_static.module.module_type);new_line;

         put(m_mod);put(module_static.module.modifier);new_line;

         put(n_instans);put(num_of_instantiations);new_line;

   end if;
end display_group_details;
```

```
  procedure display_module_details(module:MODULE_ID;EOL:in out BOOLEAN) is
begin
  EOL:=FALSE;
  get_module_info(module,
        module_static,module_dynamic,
        num_of_instantiations,EOL);
  if not EOL and module_is_alive(module) then
        new_line;
        put("Module Details:");new_line(2);

        put(present);put(module_dynamic.id.g_num);
        put(" , ");put(module_dynamic.id.l_num);new_line;
        put(l_action);put(module_dynamic.action);new_line;

        put("Child is executing?: ");put(module_dynamic.child_info.is_executing);
        put(" at Probe: ");put(module_dynamic.child_info.at_probe,4);new_line;
  end if;
end display_module_details;

  procedure display_probe_stats(probe:P_ID; EOL:in out BOOLEAN) is

length, entry_count: NATURAL;
parent_module:MODULE_ID;
task_called_as:STRING_REC;

begin

if probe_is_alive(probe) then

  get_probe_info(probe,probe_static,probe_dynamic,
        present_module,total_entry_calls,
        max_calls_in_Q,EOL);

  if  (probe_static.action= start_rendezvous) then
        new_line;
        put("ENTRY statistics for probe: ");put(probe,4);
        put(" (");put(probe_static.probe_name.s);
        put("), in Module: ( ");put(present_module.g_num,3);
        put(",");put(present_module.l_num,3);put_line(" )");
        new_line;
        put("Total of ENTRY calls:");
        put(total_entry_calls);new_line;
        put("Maximum ENTRY queue length:");
        put(max_calls_in_Q);new_line;

        reset_probe_stat_list(probe,length,EOL);
```

```
        if EOL then
          put_line("No further statistics at this time.");
        else
          for i in 1..length loop
              read_next_probe_stat(probe,
                parent_module,task_called_as,entry_count,EOL);
              if not EOL then
                new_line;
                put(parent);put(parent_module.g_num,3);
                put(",");put(parent_module.l_num,3);new_line;

                put(called_as);put_line(task_called_as.s);

                put("Entry count:");
                put(entry_count);new_line;

              end if;-- at End of list
          end loop;
        end if;-- for EOL
    end if;-- making sure probe keeps stats in the first place
  else
    EOL:=TRUE;-- Here means probe is not alive, but there may be more

  end if;-- probe is even alive

  end display_probe_stats;

  end user_interface_support;
```

```
--
-- ****************************************
-- Portable Ada Multitasking Analyzer System
--
--         Version 1.0
--
--  Designed, developed, and written by:
--         David O. LeVan
--         Robert J. Knapper
--            of the
-- Computer Software and Engineering Division
--    Institute for Defense Analyses
--         Alexandria, VA
--
--           11/8/88
--
-- ****************************************
--
with text_io;use text_io;
with user_interface_support;use user_interface_support;
with mtd_complex_types;use mtd_complex_types;
with useful_generics;use useful_generics;
with new_a_strings;use new_a_strings;
with user_interface_types;use user_interface_types;
with mtd_fundamental_types;use mtd_fundamental_types;

package user_interface is

procedure user_action(
  link_flag:in out FLAG;
  probe_flag:in out FLAG;
  signal_flag:in out FLAG);
--
-- This is the user interface module, currently implemented as a
-- procedure called by the monitor task
--
-- link_flag: activity of link logins
-- probe_flag: activity of probe logins
-- signal_flag: activity of probes signalling to monitor
--

end user_interface;
```

```
--
-- ******************************************
-- Portable Ada Multitasking Analyzer System
--
--          Version 1.0
--
--  Designed, developed, and written by:
--          David O. LeVan
--          Robert J. Knapper
--             of the
-- Computer Software and Engineering Division
--     Institute for Defense Analyses
--          Alexandria, VA
--
--             11/8/88
--
-- ******************************************
--
with mtd_data_control;use mtd_data_control;

package body user_interface is

package int_io is new integer_io(INTEGER);use int_io;

package action_enum is new enumeration_io(MTD_ACTION);use action_enum;

package check_enum is new enumeration_io(PROBE_CONDITION);
  use check_enum;

package break_enum is new enumeration_io(BREAK_PT);use break_enum;

package tasking_action_enum is new enumeration_io(TASK_ACTION);
use tasking_action_enum;

package probe_state_enum is new enumeration_io(STATE_CONTROL);
use probe_state_enum;

package fix_io is new fixed_io(DURATION);use fix_io;

        status_error:   exception;
        mode_error:             exception;
        name_error:             exception;
        use_error:              exception;
        device_error:   exception;
        end_error:              exception;
        data_error:             exception;
        layout_error:   exception;
```

```
verbose:BOOLEAN:=FALSE;-- controls amount of details printed to user

first_time_thru: BOOLEAN:=TRUE;

io_err:BOOLEAN:=FALSE;

procedure print_condition(
  data: PROBE_INFO) is

begin
  new_line;
  put("Module:");put(data.module.g_num);put(",");
       put(data.module.l_num);new_line;
  put("Name:");put_line(data.name.s);
  put("Probe ID:");put(data.id);new_line;
  put("Task Action:");put(data.tasking_action);new_line;
  put("Condition action:");put(data.action);new_line;
  put("Check what?:");put(data.condition);new_line;
  if data.action=break then
       put("Break point status:");put(data.break_status);new_line;
  elsif data.action=control then
       put("Probe Control Action: ");put(data.control);new_line;

       case data.control is
       when raise_exception=>

         case data.control_info.sel_except is
         when 1=>
              put_line("1. CONSTRAINT_ERROR.");
         when 2=>
              put_line("2. NUMERIC_ERROR.");
         when 3=>
              put_line("3. PROGRAM_ERROR.");
         when 4=>
              put_line("4. STORAGE_ERROR.");
         when 5=>
              put_line("5. TASKING_ERROR");
         when 6=>
              put_line("6. No Exceptions.");
         when others=>
              null;
         end case;-- selected exception

         when delay_task =>
              put("Delay value: ");put(data.control_info.delay_val);
              new_line;
```

```
            when others=>
                null;
        end case;-- control

    end if;-- for data.action
end print_condition;

procedure input_probe_cntl(
    probe_action:in out STATE_CONTROL;
    probe_control:in out STATE_CONTROL_INFO;
    io_err:in out BOOLEAN) is

ans:INTEGER;

begin

        put_line("Enter control information for probe.");
        begin -- exception block

            io_err:=FALSE;
            put("Enter action:");get(probe_action);new_line;

            exception
            when others=>
              io_err:=TRUE;
              put_line("Invalid enum type!");
        end; -- exception block
        if not io_err then
            probe_control.sel_except:=6;
            probe_control.delay_val:=DURATION(1);

            case probe_action is

            when nop=>
              null;

            when terminate_task=>
              null;

            when raise_exception=>

              loop
                  put_line("Select standard exception to raise.");
                  new_line;
                  put_line("1. CONSTRAINT_ERROR.");
                  put_line("2. NUMERIC_ERROR.");
                  put_line("3. PROGRAM_ERROR.");
```

```
                    put_line("4. STORAGE_ERROR.");
                    put_line("5. TASKING_ERROR");
                    put_line("6. No Exceptions.");
                    put("Section:");get(ans);new_line;
                    exit when ans in 1..6;
                end loop;
                if ans=6 then
                    probe_action:=nop;-- changed mind apparently
                else
                    probe_control.sel_except:=ans;
                end if;

            when delay_task=>
                loop
                    put("Enter delay value:");get(ans);new_line;

                    exit when ans >= 0;
                end loop;

                probe_control.delay_val:=DURATION(ans);
            end case;
        end if;-- not IO error on enum type

end input_probe_cntl;


procedure user_editor is

condition_info,t_condition_info:PROBE_INFO;
N_condition:INTEGER:=0;
found,error,EOL:BOOLEAN:=FALSE;
ans_1,g_num,l_num,id,ans,length:INTEGER:=0;
s_ans:STRING(1..80);
probe_static:PROBE_INFO_S;
probe_dyn:PROBE_INFO_D;
present_mod:MODULE_ID;
last:NATURAL;
max_calls_in_Q,total_entry_calls:NATURAL:=0;
look_for_task_act:TASK_ACTION:=end_of_program;


begin

loop
  begin --block for io exception
  loop
      put_line("1. LIST probes for Report.");
```

```
        put_line("2. LIST probes for Breakpointing.");
        put_line("3. LIST probes for Control.");
        put_line("4. ADD probes for Report.");
        put_line("5. ADD probes for Breakpointing.");
        put_line("6. ADD probes for Control.");
        put_line("7. DELETE probes for Report.");
        put_line("8. DELETE probes for Breakpointing.");
        put_line("9. DELETE probes for Control.");
        put_line("10. EXIT.");
        new_line;put("Selection:");
        get(ans);new_line;
        exit when ans in 1..10;
    end loop;
--
-- set defaults for condition info
--
    condition_info.module:=(-1,-1);
    condition_info.name:=to_a(" ");
    condition_info.id:=-1;
    condition_info.tasking_action:=end_of_program;
    condition_info.action:=report;
    condition_info.condition:=check_name;
    condition_info.active:=TRUE;
    condition_info.break_status:=reset;
    condition_info.control:=nop;
    condition_info.control_info:=(6,DURATION(1));

    case ans is
    when 1 | 2 | 3 =>

        l_num:=0;
        reset_condition_list(length);

        loop
          read_next_condition(condition_info,EOL);
          exit when EOL;
          l_num:=l_num+1;
          if condition_info.active then

              if(condition_info.action=report) and
                (ans=1) then
                new_line;
                put("Report Condition ");put(l_num);new_line;
                print_condition(condition_info);
              elsif (condition_info.action=break) and
                (ans=2) then
                new_line;
```

```
                put("Break Condition ");put(l_num);new_line;
                print_condition(condition_info);
              elsif (condition_info.action=control) and
                (ans=3) then
                new_line;
                put("Control Condition ");put(l_num);new_line;
                print_condition(condition_info);
              end if;
          end if;
        end loop;

when 4 | 5 | 6 =>

loop
      begin -- another block for io exception
      loop
        new_line;
        put_line("1. Specify by probe ID number.");
        put_line("2. Specify by probe name.");
        put_line("3. Specify by module ID.");
        put_line("4. Specify by task action.");
        put_line("5. EXIT.");
        new_line;
        get(ans_1);new_line;skip_line;
        exit when ans_1 in 1..5;
      end loop;

      error:=FALSE;
      case ans_1 is

      when 1 =>

        put("Enter probe ID:");

        get(id);new_line;skip_line;

            get_probe_info(id,
              probe_static,probe_dyn,
              present_mod,
              total_entry_calls,max_calls_in_Q,EOL);
            error:=EOL;
            if not EOL then
              condition_info.module:=present_mod;
              condition_info.name:=probe_static.probe_name;
              condition_info.id:=id;
              condition_info.condition:=check_id;
            else
```

```
            put_line("Non-existent probe ID!!");
          end if;

when 2 =>

   put("Enter probe's name:");
   get_line(s_ans,last);
   if last >0 then

          put("Enter group containing the probe:");
          get(g_num);new_line;skip_line;

          condition_info.module:=(g_num,-1);

          condition_info.name:=(POSITIVE(last),s_ans(1..last));

          condition_info.id:=get_probe_id(
            g_num,condition_info.name);

          if condition_info.id =-1 then
            condition_info.condition:=check_name;
          else
            condition_info.condition:=check_id;
            get_probe_info(condition_info.id,
                  probe_static,probe_dyn,
                  present_mod,
                  total_entry_calls,max_calls_in_Q,EOL);
            condition_info.module:=present_mod;
          end if;

   else
          error:=TRUE;
   end if;

when 3=>

   put("Enter group:");get(g_num);
   put("Enter link # (-1 for any in group):");
   get(l_num);new_line;skip_line;
   condition_info.module:=(g_num,l_num);
   condition_info.condition:=check_module;

when 4=>

   put("Enter task action (Enumeration type):");
   get(look_for_task_act);new_line;skip_line;
   condition_info.tasking_action:=look_for_task_act;
```

```
            condition_info.condition:=check_task_action;

        when others =>
            exit;
        end case;

    if ans=5 then
            condition_info.action:=break;
        elsif ans=6 then
            condition_info.action:=control;
            input_probe_cntl(
                condition_info.control,condition_info.control_info,error);
    end if;

        if not error then

            create_condition(condition_info);

        end if;

exception
        when status_error=>
            put_line("*** STATUS_ERROR exception.");

        when mode_error=>
            put_line("*** MODE_ERROR exception.");

        when name_error=>
            put_line("*** NAME_ERROR exception.");

        when use_error=>
            put_line("*** USE_ERROR exception.");

        when device_error=>
            put_line("*** DEVICE_ERROR exception.");

        when end_error=>
            put_line("*** END_ERROR exception.");

        when data_error=>
            put_line("*** DATA_ERROR exception.");

        when layout_error=>
            put_line("*** LAYOUT_ERROR exception.");

        when others=>
            put_line("*** GENERAL IO exception.");
```

```
        skip_line;

    end;-- for exception
end loop;

when 7 =>

    put("Enter condition number to delete:");get(ans_1);
    new_line;skip_line;

    remove_nth_condition(ans_1,REPORT,EOL);

when 8 =>

    put("Enter condition number to delete:");get(ans_1);
    new_line;skip_line;

    remove_nth_condition(ans_1,BREAK,EOL);

when 9=>

    put("Enter condition number to delete:");get(ans_1);
    new_line;skip_line;

    remove_nth_condition(ans_1,CONTROL,EOL);

when others =>

    exit;

end case;

exception
    when status_error=>
      put_line("*** STATUS_ERROR exception.");

    when mode_error=>
      put_line("*** MODE_ERROR exception.");

    when name_error=>
      put_line("*** NAME_ERROR exception.");

    when use_error=>
      put_line("*** USE_ERROR exception.");

    when device_error=>
      put_line("*** DEVICE_ERROR exception.");
```

```
        when end_error=>
          put_line("*** END_ERROR exception.");

        when data_error=>
          put_line("*** DATA_ERROR exception.");

        when layout_error=>
          put_line("*** LAYOUT_ERROR exception.");

        when others=>
          put_line("*** GENERAL IO exception.");
          skip_line;

  end;-- block for exception
end loop;
end user_editor;


procedure user_action(
  link_flag:in out FLAG;
  probe_flag:in out FLAG;
  signal_flag:in out FLAG) is


menu_sel, ans:NATURAL;
group_scope,this_group:INTEGER;
module_scope:INTEGER;
EOL:BOOLEAN:=FALSE;
condition_result:PROBE_INFO;
s_ans:STRING(1..10);
last:NATURAL;
length,id,n_steps:INTEGER:=0;
present,go_forever:BOOLEAN:=FALSE;
something_in_list:BOOLEAN:=FALSE;
group_start,group_end,link_start,link_end:NATURAL;
probe:INTEGER;
module_static:MODULE_INFO_S;
num_of_instantiations:INTEGER;

probe_action: STATE_CONTROL:=nop;
probe_control: STATE_CONTROL_INFO:=(6,DURATION(1));

begin

if first_time_thru then
  first_time_thru:= FALSE;
new_line(2);
```

```
put_line("        Portable Ada Multitasking Analyzer System");
put_line("              Version 1.0");
new_line;
put_line("--");
put_line("--");
put_line("--       DISCLAIMER OF WARRANTY AND LIABILITY");
put_line("--");
put_line("--");
put_line("--  THIS IS EXPERIMENTAL PROTOTYPE SOFTWARE.  IT IS PROVIDED ""AS IS"" ");
put_line("--  WITHOUT WARRANTY OR REPRESENTATION OF ANY KIND.  THE INSTITUTE");
put_line("--  FOR DEFENSE ANALYSES (IDA) DOES NOT WARRANT, GUARANTEE, OR MAKE");
put_line("--  ANY REPRESENTATIONS REGARDING THIS SOFTWARE WITH RESPECT TO");
put_line("--  CORRECTNESS, ACCURACY, RELIABILITY, MERCHANTABILITY, FITNESS FOR");
put_line("--  A PARTICULAR PURPOSE, OR OTHERWISE.");
put_line("--");
put_line("--  USERS ASSUME ALL RISKS IN USING THIS SOFTWARE.  NEITHER IDA NOR");
put_line("--  ANYONE ELSE INVOLVED IN THE CREATION, PRODUCTION, OR DISTRIBUTION");
put_line("--  OF THIS SOFTWARE SHALL BE LIABLE FOR ANY DAMAGE, INJURY, OR LOSS");
put_line("--  RESULTING FROM ITS USE, WHETHER SUCH DAMAGE, INJURY, OR LOSS IS");
put_line("--  CHARACTERIZED AS DIRECT, INDIRECT, CONSEQUENTIAL, INCIDENTAL,");
put_line("--  SPECIAL, OR OTHERWISE.");
put_line("--");
put_line("--");

put("Press RETURN key to continue:");get_line(s_ans,last);
end if;


if verbose then
  if link_flag.active then
       put("Link login activity:");put(link_flag.count);
       put_line(" times.");
       link_flag.active:=FALSE; link_flag.count:=0;
  end if;
  if probe_flag.active then
       put("Probe login activity:");put(probe_flag.count);
       put_line(" times.");
       probe_flag.active:=FALSE;probe_flag.count:=0;
  end if;
  if signal_flag.active then
       put("Signal to monitor activity:");put(signal_flag.count);
       put_line(" times.");
       signal_flag.active:=FALSE;signal_flag.count:=0;
  end if;
end if;

reset_condition_list(length);
```

```
for i in 1..length loop
  read_nth_condition(i,
        condition_result,EOL);
  if condition_result.active then
        something_in_list:=TRUE;
  end if;
  if condition_result.active and
        (condition_result.action=report)
        and probe_is_waiting(condition_result.id) then
        display_probe_details(condition_result.id,EOL);
        present:=TRUE;
  end if;
end loop;

for i in 1..length loop
  read_nth_condition(i,condition_result,EOL);
  if condition_result.active and
        (condition_result.action=break)
        and probe_is_waiting(condition_result.id) then
        display_probe_details(condition_result.id,EOL);
        present:=TRUE;
  end if;
end loop;

if  (length=0) or present or (not something_in_list) then

  loop
        begin-- for exception block
        loop
          put_line("1. Display groups.");
          put_line("2. Display all probes.");
          put_line("3. Display waiting probes.");
          put_line("4. Display modules for a group.");
          put_line("5. Display probes for a module.");
          put_line("6. Display probe details.");
          put_line("7. Trace.");
          put_line("8. Edit the condition list.");
          put_line("9. Display trapped probes.");
          put_line("10.Release a probe.");
          put_line("11.Monitor setups.");
          put_line("12. Display Task Entry statistics.");
          put_line("13. Set Probe Controls.");
          put_line("14. Exit.");
          new_line;put("Enter selection:");
          get(menu_sel);new_line;
          skip_line;
```

```
      exit when menu_sel in  1..14;
    end loop;

    case menu_sel is
    when 1 =>
      display_groups;

    when 2 =>
      display_all_probes;

    when 3 =>
      display_waiting_probes;

    when 4 =>

          loop

            new_line;
            put("Enter module ID; Group (-1 indicates all, 0 for EXIT):");
            get(group_scope);new_line;
            exit when group_scope=0;

            put("Link number (-1 indicates all links for Group):");
            get(module_scope);new_line;skip_line;
--
-- Set up loops to index through the desired groups and links
--
            reset_module_group_list(group_end);

            loop

              if group_scope =-1 then
                read_next_group(this_group,module_static,
                      num_of_instantiations);
                exit when this_group= -1;
              else
                this_group:=group_scope;
              end if;


              if module_scope=-1 then
                link_start:=1;
                reset_link_list(this_group,link_end,EOL);
              else
                link_start:=module_scope;
                link_end:=module_scope;
              end if;
```

```
                for present_link in link_start..link_end loop
                  display_module_details(
                        (this_group,present_link),EOL);
                end loop;-- present link
              exit when group_scope /= -1;
              end loop;-- present group

            end loop;-- display module info mode



when 5 =>
  put_line("Enter module ID");
  put("Group:");get(group_scope);
  put("  Module number:");get(module_scope);
  new_line;
  display_module_probes(
       (group_scope,module_scope), EOL);
  if EOL then
       put_line("Invalid Module ID!");
  end if;

when 6 =>
  loop
       put("Enter probe ID (0 to exit):");
       get(id);new_line;
       exit when id=0;
       display_probe_details(id,EOL);
       if EOL then
         put_line("Invalid probe ID entered!");
       end if;
  end loop;

when 7 =>
  loop
       put("Enter number of steps to trace:");
       get(n_steps);new_line;
       exit when n_steps>0;
  end loop;

when 8 =>

  user_editor;

when 9=>
  present:=FALSE;
  for i in 1..length loop
```

```
  read_nth_condition(i,
        condition_result,EOL);
  if condition_result.active and
        (condition_result.action=break)
        and (condition_result.break_status=tripped) then
          display_probe_details(condition_result.id,EOL);
          present:=TRUE;
  end if;
  end loop;
  if not present then
        put_line("No probes trapped at this time.");new_line;
  end if;

when 10 =>

  put("Enter probe ID to release:");get(id);
  new_line;skip_line;
  present:=FALSE;
  for i in 1..length loop
        read_nth_condition(i,
          condition_result,EOL);
        if condition_result.active and
          (condition_result.action=break)
          and (id=condition_result.id) then
          condition_result.break_status:=reset;
          write_nth_condition(i,
                condition_result,EOL);
          present:=TRUE;
          exit;
        end if;
  end loop;
  if not present then
        put_line("No such probe to release!!");
  end if;

when 11 =>

loop
  loop
        put_line("1. Toggle Verbose mode.");
        put_line("2. EXIT setup mode.");
        get(ans);new_line;skip_line;
        exit when ans in 1..2;
  end loop;

  case ans is
  when 1 =>
```

```
                verbose:=not verbose;
                if verbose then
                  put_line("Verbose mode on.");
                else
                  put_line("Verbose mode off.");
                end if;
              when others =>
                exit;
              end case;

            end loop;

            when 12 =>

            loop
                new_line;
                put("Enter module ID; Group (-1 indicates all, 0 for EXIT):");
                get(group_scope);new_line;
                exit when group_scope=0;

                put("Link number (-1 indicates all links for Group):");
                get(module_scope);new_line;skip_line;
--
-- Set up loops to index through the desired groups and links
--
                if group_scope =-1 then
                  group_start:=1;
                  reset_module_group_list(group_end);
                else
                  group_start:=group_scope;
                  group_end:=group_scope;
                end if;

                for present_group in group_start..group_end loop
                  if module_scope=-1 then
                      link_start:=1;
                      reset_link_list(present_group,link_end,EOL);
                  else
                      link_start:=module_scope;
                      link_end:=module_scope;
                  end if;

                  for present_link in link_start..link_end loop
                      reset_link_probe_list((present_group,present_link),
                        length,EOL);
                      loop
                        read_next_link_probe(
```

```
                        (present_group,present_link),probe);
                    exit when probe =-1;-- No more entries
                    display_probe_stats(probe,EOL);
                  end loop;-- Does all the real work
                end loop;-- present link
              end loop;-- present group

        end loop;-- display stats mode

        when 13=>

        loop
          loop
              put("Enter ID of Probe (0 to EXIT):");
              get(probe);new_line;
              exit when probe=0;
              if not probe_is_alive(probe) then
                put_line("Invalid Probe ID!");-- making sure ID is good
              else
                exit;
              end if;
          end loop;

          exit when probe=0; -- that's all for now

          put_line("Probe selected:");
          display_probe_details(probe,EOL);new_line;

-- Get Probe's control information

          input_probe_cntl(probe_action,probe_control,EOL);
          if not EOL then
              set_probe_control(probe,probe_action,probe_control,EOL);
          end if;
        end loop;-- main one for probe control

        when others =>
          exit;-- from main menu

        end case;-- select from main menu

    exception
        when status_error=>
          put_line("*** STATUS_ERROR exception.");

        when mode_error=>
          put_line("*** MODE_ERROR exception.");
```

```
            when name_error=>
              put_line("*** NAME_ERROR exception.");

            when use_error=>
              put_line("*** USE_ERROR exception.");

            when device_error=>
              put_line("*** DEVICE_ERROR exception.");

            when end_error=>
              put_line("*** END_ERROR exception.");

            when data_error=>
              put_line("*** DATA_ERROR exception.");

            when layout_error=>
              put_line("*** LAYOUT_ERROR exception.");

            when others=>
              put_line("*** GENERAL IO exception.");
              skip_line;

            end;-- exception block
        end loop;
    end if;

        exception
        when DATA_ERROR=>
            put_line("General I/O error raised!! in USER_INTERFACE package");
        when others=>
            put_line("General exception raised in USER_INTERFACE package");
    end user_action;


    end user_interface;
```

```
--
-- ****************************************
-- Portable Ada Multitasking Analyzer System
--
--          Version 1.0
--
--  Designed, developed, and written by:
--          David O. LeVan
--          Robert J. Knapper
--             of the
-- Computer Software and Engineering Division
--    Institute for Defense Analyses
--          Alexandria, VA
-
--             11/8/88
--
-- ****************************************
--
with useful_types; use useful_types;
with new_a_strings; use new_a_strings;
with mtd_fundamental_types; use mtd_fundamental_types;
with mtd_complex_types; use mtd_complex_types;
with mtd_link; use mtd_link;

package mtd_tool is
--
-- This package spec. contains the procedures, functions, and task  that form
-- the monitor portion and probe for the multi-tasking debugger tool.
--

task mtd_monitor is
--
-- This is the multi-tasking debugger monitor task
--
   entry probe_login(
         name: STRING_REC;
         action: TASK_ACTION;
         path:STRING_REC;
         module: MODULE_ID;
         id: out P_ID);
   --
   -- Logs the probe into the monitor system.
   --
   -- name: probe's name
   -- action: Tasking operation the probe monitors activity of
   -- path: scoping path from the link tasks definition down to the
   --       probe's placement at instrumentation time.
```

```
  -- module: ID of logical module containing the probe
  -- id: ID given to probe by the monitor
  --
  entry link_login(
        static: MODULE_INFO_S;
        dynamic: in out MODULE_INFO_D);
  --
  -- Logs the logical module (Link) into the monitor system.
  --
  -- static: Static information about the module
  -- dynamic: Dynamic info.  Includes a link ID assigned at this time.
  --

  entry signal_to_monitor (
        probe_id: P_ID;
        parent_module: MODULE_ID;
        task_call_name: STRING_REC;
        number_queued:NATURAL:=0);
  --
  -- The probes signal to the monitor via this entry.
  --
  -- probe_id: unique ID assigned by monitor at login time
  -- parent_module: for post accept staements, ID of parent thread module
  -- task_call_name: for post accept statements, actual name task called with
  -- number_queued: for post accepts, number of calls currently queued on entry
  --
end mtd_monitor;



generic
  p_name: STRING;
  t_action: TASK_ACTION;
  p_path:STRING;
--
-- These generic parameters are assigned at instrumentation time:
-- p_name: name assigned to probe
-- t_action: task action probe monitors
-- p_path: scoping path to probe from the definition of the link task
--

  procedure probe(
    link_task: A_LINK;
        module: in out  MODULE_ID;
        id:in out P_ID;
        first_time:in out BOOLEAN;
        parent_module: MODULE_ID:=(-1,-1);
```

```
        task_call_name: STRING_REC:= to_a(" ");
        number_queued:NATURAL:=0);
    --
    -- This is the procedure implementing the probe operation.
    -- All pertenant info is initialized
    -- into internal data structures via the generic parameters
    --
    -- link_task: Pointer to the link task to use to communicate with probe
    -- module: Module probe is in
    -- id: Probe's ID
    -- first_time: Used to initiate a probe login action the first time called
    -- parent_module: ID of calling module for post accept probes only
    -- task_call_name: Name task actually called as, post accept only
    -- number_queued: Number of calls in queue, for post accepts only
    --

end mtd_tool; -- End of the package
```

```
--
-- ******************************************
-- Portable Ada Multitasking Analyzer System
--
--          Version 1.0
--
--  Designed, developed, and written by:
--          David O. LeVan
--          Robert J. Knapper
--             of the
-- Computer Software and Engineering Division
--     Institute for Defense Analyses
--          Alexandria, VA
--
--             11/8/88
--
-- ******************************************
--
with mtd_data_control;
use mtd_data_control;
with user_interface_types;use user_interface_types;
with user_interface;
with text_io;use text_io;


package body mtd_tool is

package int_io is new integer_io(INTEGER);use int_io;

procedure is_on_cond_list(
   probe_id: P_ID;
   condition:in out NATURAL;
   condition_info:in out PROBE_INFO;
   EOL:in out BOOLEAN) is

   probe_static: PROBE_INFO_S;
   probe_dynamic: PROBE_INFO_D;
   present_module: MODULE_ID;

   condition_length: INTEGER:=0;


   found:BOOLEAN:=FALSE;
   entries,calls_in_Q:NATURAL:=0;

begin
```

```
--
-- Check to see if the probe is on the condition trigger list.
--
  reset_condition_list(condition_length);
  found:=FALSE;
  condition:=0;

  for i in 1..condition_length loop

      read_nth_condition(i,
        condition_info,EOL);
      if condition_info.active then

        get_probe_info(probe_id,
            probe_static,probe_dynamic,
            present_module,entries,calls_in_Q,EOL);

        case condition_info.condition is

            when check_id=>
            if (probe_id=condition_info.id)  then
              found:=TRUE;
              condition:=i;
              exit;
            end if;-- probe id match

            when check_name =>
            if eq_string(condition_info.name,probe_static.probe_name) and
                  (condition_info.module.g_num=present_module.g_num) then
              found:=TRUE;
              condition_info.module:=present_module;-- fill in matching probe's info
              condition_info.id:=probe_id;
              condition_info.tasking_action:=probe_static.action;
              condition:=i;
              write_nth_condition(
                  i,
                  condition_info,EOL);
              exit;
            end if;-- name match

            when check_module =>
            if (present_module.g_num=condition_info.module.g_num) and
              ((present_module.l_num=condition_info.module.l_num) or
              (condition_info.module.l_num=-1)) then
                  found:=TRUE;
                  condition:=i;
-- Fill in the information for the probe that satisfies the wild card
```

```
                    condition_info.name:=probe_static.probe_name;
                    condition_info.id:=probe_id;
                  condition_info.tasking_action:=probe_static.action;

                    write_nth_condition(
                      i,
                      condition_info,EOL);
                    exit;
                end if;-- module match

                when check_task_action =>
                if (probe_static.action=condition_info.tasking_action) then
                  found:=TRUE;
                  condition:=i;
                  condition_info.module:=present_module;
                  condition_info.id:=probe_id;
                  condition_info.name:=probe_static.probe_name;

                  write_nth_condition(
                      i,
                      condition_info,EOL);
                  exit;
                end if;-- task action match

                end case;-- on match condition
            end if;-- condition active
        end loop;-- for condition list

-- End of List indicates no match
  EOL:= not found;-- pass on the joy of discovery

end is_on_cond_list;

task body mtd_monitor is

  probe_static: PROBE_INFO_S;
  probe_dynamic: PROBE_INFO_D;
  module_static: MODULE_INFO_S;
  module_dynamic: MODULE_INFO_D;
  present_module: MODULE_ID;
  present_probe: P_ID;
  module_group: G_ID;
  num_of_instantiations: NATURAL;
  at_eof: CONSTANT INTEGER:= -1;
  present_probe_name, present_file_name: STRING_REC;
  present_module_name: LOGICAL_MODULE;
  shut_down_all: BOOLEAN:= FALSE;
```

```
   link_flag,probe_flag,signal_flag:FLAG;
   status_of_link:LINK_STATUS:=parent_ok;

   EOL:BOOLEAN:=FALSE;
   length:NATURAL:=0;
   group_length,module_length:INTEGER:=0;
   condition_info:PROBE_INFO;
   N_condition,condition_length:INTEGER:=0;
   found,update:BOOLEAN:=FALSE;
   entries,calls_in_Q:NATURAL:=0;

   probe_control:STATE_CONTROL;
   probe_control_info:STATE_CONTROL_INFO;

begin


loop

select
   accept link_login(
         static: MODULE_INFO_S;
         dynamic: in out MODULE_INFO_D) do

   link_flag.active:=TRUE;
   link_flag.count:=link_flag.count+1;

-- create module entry
         create_module_entry(static,dynamic);

   end link_login;
or
   accept probe_login(
         name: STRING_REC;
         action: TASK_ACTION;
         path:STRING_REC;
         module: MODULE_ID;
         id: out P_ID) do

   probe_flag.active:=TRUE;
   probe_flag.count:=probe_flag.count+1;

-- create probe entry
         create_probe_entry(name,action,path,module,present_probe);
         id:= present_probe;
```

```
   end probe_login;
or
   accept signal_to_monitor(
         probe_id: P_ID;
         parent_module: MODULE_ID;
         task_call_name: STRING_REC;
         number_queued: NATURAL:=0) do

   signal_flag.active:=TRUE;
   signal_flag.count:=signal_flag.count+1;

-- set probe to wait
         set_probe_wait(probe_id,TRUE, EOL);

-- set parent information for probe
         set_parent_info(probe_id,
           parent_module,task_call_name,
           number_queued, EOL);

-- see if probe is on the condition action list
   is_on_cond_list(probe_id,
         N_condition,condition_info,EOL);
   if not EOL then

         if (condition_info.action=break) then
           condition_info.break_status:=tripped;
         end if;

         write_nth_condition(N_condition,condition_info,EOL);
   end if;-- condition found

   end signal_to_monitor;

or
   delay 1.0;

end select;

-- check for user interface info

   user_interface.user_action(
         link_flag,probe_flag,signal_flag);



-- scan probe list to find all waiting probes, pass control information
-- to them while releasing them
```

```
-- reset probe list
reset_probe_list(length);

for i in 1..length
loop


-- if probe is waiting then get its module information and =>
  if probe_is_waiting(i) then

  get_probe_info(i,
       probe_static,probe_dynamic,
       present_module,entries,calls_in_Q,EOL);
  found:=FALSE;
--
-- check to see if probe is on breakpoint list.  If found, then
-- don't release it later
--
  reset_condition_list(condition_length);

  for j in 1..condition_length loop
       read_nth_condition(j,
         condition_info,EOL);
       if (i=condition_info.ID) and
         (condition_info.action=break) and
         condition_info.active and
         (condition_info.break_status=tripped)  then
         found:=TRUE;
         exit;
       end if;
  end loop;

  if not found then

-- release the probe

  get_module_info(present_module,
       module_static,module_dynamic,
       num_of_instantiations, EOL);


  case probe_static.action is
  when task_terminate | task_abort | task_end =>
       module_dynamic.action:=pass_info_terminate;

  when others =>
       module_dynamic.action:= pass_info;
```

```
  end case;

  is_on_cond_list(i,N_condition,condition_info,EOL);

  if EOL then
-- not on condition list, use probe's control info
       probe_control:=probe_dynamic.action;
       probe_control_info:=probe_dynamic.control_info;

  elsif  condition_info.action=control then
-- pass on control info from condition list
       probe_control:=condition_info.control;
       probe_control_info:=condition_info.control_info;
  end if;

  status_of_link:=parent_ok; -- we assume

if module_dynamic.link_task'CALLABLE then

  select
       module_dynamic.link_task.signal_from_monitor(
         probe_control,
         probe_control_info,
         module_dynamic.action, status_of_link);


    if status_of_link=parent_not_there then
-- link task had a problem rendezvousing with its parent
         put("Parent Not There for Module:");put(present_module.g_num);
         put(" , ");put(present_module.l_num);new_line;
       else
--  set probe's wait to FALSE

         set_probe_wait(i,FALSE, EOL);
--
-- check to see if normal end of program has occured
--
         shut_down_all:=(probe_static.action=end_of_program);
       end if;

  else
       put("Signal_from_monitor rendezvous cannot be made for Module:");put(present_module.g_num);
         put(" , ");put(present_module.l_num);new_line;

  end select;
else
  put("Link not present for Module:");put(present_module.g_num);
```

```
            put(" , ");put(present_module.l_num);new_line;

   for countdown in 1..10 loop
         if module_dynamic.link_task'CALLABLE then

            put("Link established for Module:");put(present_module.g_num);
            put(" , ");put(present_module.l_num);new_line;

            exit;
         else
            delay 1.0;

         end if;
   end loop; -- for countdown
end if; -- for CALLABLE

   case probe_static.action is

   when task_terminate | task_abort | task_end =>

-- clean up after module

      remove_module_entry(module_dynamic.id, EOL);

   when others =>
         null;
   end case;

   end if;-- for if not found

   end if;-- for if probe waiting

end loop;-- for i

-- if shut down all then do it
if shut_down_all then
   new_line(2);
   put_line("Normal end of program, SHUTTING DOWN!!");

--
-- Move through the lists of groups and the links associated with each
-- terminating each link task in turn.
--
   reset_module_group_list(group_length);

   for group in 1..group_length loop
```

```
        read_next_group(module_group,module_static,
          num_of_instantiations);
        reset_link_list(module_group,module_length,EOL);

        for module in 1..module_length loop

          read_next_link(module_group,module_dynamic,EOL);
          if not EOL then
                put("Terminating link for module:");put(module_group);
                put(" , ");put(module);new_line;

                status_of_link:=parent_ok;

                if module_dynamic.link_task'CALLABLE then
                select
                  module_dynamic.link_task.signal_from_monitor(
                        proc_action=>nop,
                        selection=>(1,DURATION(1.0)),
                        l_action=>terminate_link,
                        l_status=>status_of_link);
                or
                  delay 1.0;
                end select;

                else
                  put_line("Link task already terminated!!");
                end if;-- for CALLABLE
            end if;-- for End of list
          end loop;-- next link
    end loop;-- next group

    put_line("Exiting the monitor system now.");
    exit;  -- leave the main loop and terminate this task

end if;

end loop; -- main loop


end mtd_monitor;

    procedure probe(
      link_task: A_LINK;
          module: in out  MODULE_ID;
          id: in out P_ID;
          first_time: in out BOOLEAN;
          parent_module: MODULE_ID:= (-1,-1);
```

```
task_call_name: STRING_REC:= to_a(" ");
number_queued:NATURAL:=0) is


action: STATE_CONTROL;
control_info: STATE_CONTROL_INFO;

begin

  if first_time then

      mtd_monitor.probe_login(
        to_a(p_name),
        t_action,
        to_a(p_path),
        module,
        id);
      first_time:= FALSE;

  end if;

  mtd_monitor.signal_to_monitor(
      id,parent_module,task_call_name,number_queued);

  link_task.signal_to_process(
      action, control_info);


  case action is

      when nop=>
        null;

      when terminate_task=>
        new_line;put("*** Probe ");put(p_name);
        put(" in ");put(p_path);
        put_line(" is TERMINATING task! ***");

        terminate;
        null;

      when raise_exception=>
        case control_info.sel_except is
            when 1=>
              new_line;put("*** Probe ");put(p_name);
              put(" in ");put(p_path);
              put_line(" is raising exception CONSTRAINT_ERROR! ***");
```

```
                raise CONSTRAINT_ERROR;
              when 2=>
                new_line;put("*** Probe ");put(p_name);
                put(" in ");put(p_path);
                put_line(" is raising exception NUMERIC_ERROR! ***");
                raise NUMERIC_ERROR;
              when 3=>
                new_line;put("*** Probe ");put(p_name);
                put(" in ");put(p_path);
                put_line(" is raising exception PROGRAM_ERROR! ***");
                raise PROGRAM_ERROR;
              when 4=>
                new_line;put("*** Probe ");put(p_name);
                put(" in ");put(p_path);
                put_line(" is raising exception STORAGE_ERROR! ***");
                raise STORAGE_ERROR;
              when 5=>
                new_line;put("*** Probe ");put(p_name);
                put(" in ");put(p_path);
                put_line(" is raising exception TASKING_ERROR! ***");
                raise TASKING_ERROR;
              when others=>
                null;
          end case;

        when delay_task=>
          delay control_info.delay_val;

    end case;


  end probe;


end mtd_tool; -- End of package body
```

```
--
-- ****************************************
-- Portable Ada Multitasking Analyzer System
--
--          Version 1.0
--
--  Designed, developed, and written by:
--          David O. LeVan
--          Robert J. Knapper
--             of the
-- Computer Software and Engineering Division
--     Institute for Defense Analyses
--          Alexandria, VA
--
--             11/8/88
--
-- ****************************************
--
with useful_types; use useful_types;
with new_a_strings; use new_a_strings;
with mtd_fundamental_types; use mtd_fundamental_types;
with mtd_complex_types; use mtd_complex_types;
with mtd_link; use mtd_link;
with mtd_tool; use mtd_tool;

generic
   m_group: G_ID:= -1;
   f_name: STRING;
   m_name: STRING;
   m_type: TYPE_OF_MODULE:= is_task;
   m_modifier: TYPE_OF_MODULE_MODIFIER:= is_normal;
--
-- These generic parameters are assigned at instrumentation time
--
-- m_group: Module group number.
-- f_name: File name containing the source of the code this module
-- is in.
-- m_name: Name of this module; file, package, or task name.
-- m_type: Task, package, or task.
-- m_modifier: Is the module a regular, generic or task type?
--
   package link_init is
--
-- This package initializes the link task and
-- setups and initialize the needed data structures.
--
   module_static_info: MODULE_INFO_S;
```

```
    module_dynamic_info: MODULE_INFO_D;

end link_init; -- End of the Generic package
```

```
--
-- ****************************************
-- Portable Ada Multitasking Analyzer System
--
--         Version 1.0
--
--  Designed, developed, and written by:
--         David O. LeVan
--        Robert J. Knapper
--           of the
-- Computer Software and Engineering Division
--    Institute for Defense Analyses
--         Alexandria, VA
--
--           11/8/88
--
-- ****************************************
--
with text_io;use text_io;

package body link_init is

probe_init_first_time: BOOLEAN:= TRUE;
probe_init_id: P_ID:= -1;

  procedure probe_init is new probe(
       p_name=>"probe_init",
       t_action=> task_init,
       p_path=>".");

begin

  module_static_info.file_name:=to_a(f_name);-- file name in which source is defined
  module_static_info.module.module_name:= to_a(m_name);-- module name
  module_static_info.module.module_type:= m_type;-- task, file, package, etc.
  module_static_info.module.modifier:= m_modifier;-- normal, type, or generic

  module_dynamic_info.link_task:= new LINK;-- create the link task
  module_dynamic_info.id:= (g_num=> m_group,l_num=> -1);-- initialize module ID
  module_dynamic_info.action:= pass_info;-- default

-- code for robust task startup wait
  for i in 1..6 loop
  if mtd_monitor'CALLABLE then

  mtd_monitor.link_login(
       module_static_info,
```

```
        module_dynamic_info);
    exit;
    else
        delay 10.0;
        put_line("Cannot perform task call to monitor: LINK_LOGIN.");
    end if;
    end loop;

--
-- Inserted probe to allow monitor to pause in package declaritive region
--
    probe_init(
      link_task=>module_dynamic_info.link_task,
        module=>module_dynamic_info.id,
        id=>probe_init_id,
        first_time=>probe_init_first_time
        );
--
-- End of inserted probe
--


end link_init;
```

Distribution List for IDA Paper P-2124

NAME AND ADDRESS	NUMBER OF COPIES
Sponsor	
Dr. John F. Kramer Program Manager STARS DARPA/ISTO 1400 Wilson Blvd. Arlington, VA 22209-2308	4
Other	
Defense Technical Information Center Cameron Station Alexandria, VA 22314	2
IDA	
General W.Y. Smith, HQ	1
Ms. Ruth L. Greenstein, CSED	1
Mr. Philip L. Major, HQ	1
Dr. Robert E. Roberts, HQ	1
Dr. Richard J. Ivanetich, CSED	1
Mr. Robert J. Knapper, CSED	1
IDA Control & Distribution Vault	2